Shape Spaces from Morphing

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)

von

Dipl.–Inform. Marc Alexa

aus Darmstadt

Referenten der Arbeit: *Prof. Dr.–Ing. J. L. Encarnação,*
Prof. Dr. Markus Gross

Tag der Einreichung: 08. März 2002
Tag der mündlichen Prüfung: 19. April 2002

D 17
Darmstädter Dissertation 2002

Zusammenfassung

In dieser Arbeit werden Methoden zur Repräsentation der Gestalt oder Form von Objekten vorgestellt. Die Grundidee ist, die Form eines Objektes als Mischung anderer vorgegebener Formen zu beschreiben. Dazu wird das mathematische Konzept linearer Räume verwendet: Einige Objekte bilden die Basis eines Raumes, und deren Kombination erzeugt die Elemente dieses Raumes.

Diese Art der Beschreibung hat zwei Vorteile gegenüber der weit verbreiteten absoluten Repräsentation: Sie ist kompakt, wenn die Anzahl der Basen klein im Vergleich zur geometrischen Komplexität der Objekte ist. Sie ist deskriptiv, wenn die Basisformen eine Semantik haben, da dann die Anteile an diesen Basisformen das Objekt beschreiben.

Zur Darstellung der Basisformen werden hier polygonale Netze verwendet. Die Arbeit beschäftigt sich daher mit der Kombination gegebener Polygonnetze und verschiedenen Anwendungen, die bei dieser Art der graphischen Modellbeschreibung auf der Hand liegen.

Die Transformation eines gegebenen Objektes in ein anderes wird in der graphischen Datenverarbeitung *Morphing* genannt. Das Ergebnis dieser Transformation kann in der hier verwendeten Terminologie als ein ein-dimensionaler Raum verstanden werden. Durch weitere Transformationen mit zusätzlichen Basisformen ergeben sich höher-dimensionale Räume. Zum gegenwärtigen Zeitpunkt sind Morphing-Verfahren für polygonale Netze wegen topologischen und geometrischen Problemen noch verbesserungsbedürftig, weshalb sich der erste Teil dieser Arbeit mit solchen Verfahren befasst.

Diese Morphing-Verfahren werden dann so erweitert, dass sie die Kombination von mehr als zwei Netzen erlauben. Die Nützlichkeit dieser Beschreibung von Gestalt wird an Hand von zwei Szenarien demonstriert: Zur Visualisierung von Multiparameter-Informationsdaten, wobei die Parameter auf Glyphen abgebildet werden und zur effizienten Speicherung und Übermittelung von geometrischen Animationen.

Übersicht

Der erste Teil dieser Arbeit widmet sich Morphing-Vefahren für polygonale Netze. Hier beschränken wir die Polygone auf Dreiecke. Ein Dreiecksnetz $\mathcal{M}$ lässt sich

wie folgt durch ein Paar (K, V) beschreiben [Spanier 1966]: Ein (abstrakter) Simplizialkomplex K beschreibt die Topologie des Netzes, und $V = (\mathbf{v_1}, \ldots, \mathbf{v_n})$ die Geometrie der Knoten in $\mathbb{R}^d$ mit typischerweise $d = 3$.

Kanten und Facetten sind in K als Paare $\{i, j\}$ und Tripel $\{i, j, k\}$ dargestellt. Die topologische Realisierung bildet K auf einen (geometrischen) Simplizialkomplex $|K|$ in $\mathbb{R}^n$ ab: Die Knoten werden mit der kanonischen Basis des $\mathbb{R}^n$ identifiziert, so dass jedes Simplex $\mathbf{s}$ durch die konvexe Hülle der Punkte $\{\mathbf{e_i}\} \in \mathbb{R}^n, i \in \mathbf{s}$ beschrieben ist. Die geometrische Realisierung $\phi_V(|K|)$ ist eine lineare Abbildung des Simplizialkomplexes $|K|$ auf $\mathbb{R}^d$, die durch die Identifikation der Einheitsvektoren $\mathbf{e_i} \in \mathbb{R}^n$ mit den Koordinaten $\mathbf{v_i} \in V$ hergestellt wird. Die Abbildung ϕ_V heisst Einbettung wenn sie bijektiv ist. Für jede Einbettung gilt, dass sich jeder Punkt $\mathbf{p}$ auf dem Netz durch eine baryzentrische Koordinate $\mathbf{b}$ mit höchstens drei nicht verschwindenden Komponenten repräsentieren lässt.

Typische Morphingverfahren für polygonale Netze beginnen mit zwei Netzen $\mathcal{M}_0 = (K_0, V_0)$ und $\mathcal{M}_1 = (K_1, V_1)$. Das Ziel des Verfahrens ist die Erzeugung einer Familie $\mathcal{M}(t) = (K, V(t)), t \in [0, 1]$ so dass die Formen $V(0)$ und $V(1)$ geometrisch mit Ausgangsformen identisch sind oder sie zu einem vorgegebenen Grad approximieren, also $\phi_{V(0)}(|K|) \approx \phi_{V_0}(|K_0|)$ und $\phi_{V(1)}(|K|) \approx \phi_{V_1}(|K_1|)$ Die Erzeugung dieser Familie lässt sich in drei Schritten durchführen:

1. Zwischen den Netzen muss eine Korrespondenz hergestellt werden. Das Ergebnis sind Koordinatenvektoren W_0, W_1, die die Positionen der Knoten auf dem anderen Netz angeben: $W_0 \in \phi_{V_1}(|K_1|)$. Jeder dieser Knoten kann dann auch mit einer baryzentrischen Koordinate in Bezug auf das andere Netz definiert werden.

2. Auf Basis der korrespondierenden Netze wird ein gemeinsames Netz K erzeugt, das beide geometrischen Instanzen über entsprechende Koordinatenvektoren $V(0), V(1)$ darstellen kann. Dies geschieht entweder durch Überlagerung der Graphen oder durch die Erzeugung eines neuen semi-regulären Netzes.

3. Für das gemeinsame Netz müssen Knotenpfade $V(t), t \in]0, 1[$ bestimmt werden, die alle Knoten vom Ursprungszustand in die gewünschte Endposition überführen.

Jedem dieser drei Schritte ist in der Arbeit ein Kapitel gewidmet, in dem bestehende Verfahren kategorisiert und eigene Beiträge vorgestellt werden.

Im darauffolgenden Kapitel wird die Erweiterung auf mehr als zwei Basisformen erörtert. Dies geschieht zunächst in abstrakter Weise und dann für den konkreten Fall von polygonalen Netzen und den existierenden Verfahren. Die abschliessenden zwei Kapitel zeigen Anwendungsmöglichkeiten in den Bereichen Informations-Visualisierung und geometrische Animation.

Korrespondenzabbildungen für polygonale Netze

Für zwei gegebene Netze $\mathcal{M}_0$ und $\mathcal{M}_1$ sollen baryzentrische Koordinaten B_0 bestimmt werden, so dass die assozierte geometrische Form $W_0 = \phi_{V_1}(B_0)$ der Koordinaten auf dem Netz $\mathcal{M}_1$ eine Einbettung ϕ_{W_0} von $\mathcal{M}_0$ auf $\mathcal{M}_1$ ist. Dabei wird angenommen, dass die Abbildung der Knoten des Netzes den wesentlichen Teil der Bijektion zwischen den Oberflächen ausmacht, was voraussetzt, dass die Netze genügend fein tesseliert sind.

Dieser Schritt wird zumeist durch die Abbildung der Fläche auf einen geeigneten gemeinsamen Parameterraum D umgesetzt. Typische Parameterräume sind die Kugel $\mathbb{S}^2$ (für den Fall, dass die Netze homöomorph zu Kugeln sind) sowie ein Atlas bestehend aus disjunkten topologischen Kreisscheiben. Eine wesentliche Nebenbedingung im Morphing ist die Berücksichtigung von vorgegebenen Punkt-zu-Punkt Korrespondenzen, die in einem Präprozess automatisch oder durch den Benutzer festgelegt werden.

Im Falle der Abbildung über die Kugel wird zunächst eine Einbettung Φ_S mit $S = \{s_0, s_1, \ldots\}, s_i \in \mathbb{R}^3, |s_i| = 1$ bestimmt, die dann durch eine bijektive Abbildung f an die Korrespondenzbedingungen angepasst wird.

$$\begin{array}{ccc} \{i\} \in K_0 & \xrightarrow{W_0} & \phi_{V_1}(B_0) \\ {\scriptstyle \phi_{S_0}}\downarrow & & \uparrow{\scriptstyle \phi_{S_1}^{-1}} \\ \mathbb{S}^2 & \xrightarrow[f]{} & \mathbb{S}^2 \end{array}$$

Bei diesem Ansatz gilt es im wesentlichen, Verfahren zur Einbettung auf der Kugel und geeignete Abbildungsfunktionen f zu bestimmen.

Die Zerlegung in Kacheln ist für eine grössere Klasse von Modellen verwendbar, aber auch komplizierter. Zusätzlich zur Einbettung der Teilnetze auf den Kreisscheiben müssen die Ausgangsnetze in isomorphe Atlanten zerlegt werden. Dazu verwendet man einen Simplizialkomplex L, der aus einer Teilmenge der Knoten von K_0, K_1 besteht:

$$\phi_{V_0}(|L|) \approx \phi_{V_0}(|K_0|), \phi_{V_1}(|L|) \approx \phi_{V_1}(|K_1|)$$

L ist (topologischer) Minor von sowohl K_0 als auch K_1. Jeder Knoten in K_0, K_1 wird mit einer Facette in L identifiziert. Der gemeinsame Parameterraum ist dann die geomtrische Realisierung $|L|$, wo jeder Knoten von K_0, K_1 als baryzentrische Koordinate repräsentiert ist.

$$\begin{array}{ccc} \{i\} \in K_0 & \xrightarrow{W_0} & \phi_{V_1}(B_0) \\ {\scriptstyle \phi_{L_0}}\downarrow & & \uparrow{\scriptstyle \phi_{L_1}^{-1}} \\ |L| & \xrightarrow[f]{} & |L| \end{array}$$

Um diese Verfahren umzusetzen, sind mithin Methoden notwendig, die Teilnetze in der Ebene parametrisieren, geeignete geschlossene Netze auf der Kugel parametrisieren und aus zwei gegeben Netzen eine gemeinsame Zerlegung in Kacheln erzeugen.

Zur Beachtung von vom Anwender spezifizierter Korrespondenzen muss man das Netz im Parameterraum mit einer geeigneten Funktion f auf sich selbst abbilden, so dass die Abstände der korrespondierenden Knoten minimiert wird. Dies kann nicht mit einer kontinuierlichen Abbildungsfunktion geschehen, weil diese u.U. die Einbettungseigenschaft der Abbildung zunichte machen würde. Man sollte daher iterativ vorgehen.

Abbildung von Teilnetzen in die Ebene

Bei der Parametrisierung von Netzen in der Ebene haben sich eine Reihe von Verfahren durchgesetzt. Die meisten dieser Verfahren beginnen damit, den Rand des Teilnetzes auf einem Kreis anzuordnen [Tutte 1963; Floater 1997; Floater 2002; Floater 2001; Pinkall & Polthier 1993; Polthier 2000; Eck et al. 1995; Gregory et al. 1998; Gregory et al. 1999]. Es gibt weitere Verfahren ([Hormann & Greiner 2000; Lévy & Mallet 1998; Lévy 2001; Zigelmann et al. 2002]), die eine solche Anordnung nicht erfordern, diese sind aber für Korrespondenzabbildung nicht besser geeignet, da ja auch eine Bijektion auf dem Rand gefordert ist.

Bei fixiertem Rand werden die Positionen der freien Knoten als Linearkombination der Nachbarn beschrieben. Die einfachste Wahl ist, jeden Knoten im Schwerpunkt seiner Nachbarn anzuordnen. Man kann den Knoten auch als gewichtete Summe der Nachbarn darstellen, wobei sich die Gewichte aus der 3D-Geometrie ergeben. Die Knotenpositionen ergeben sich letztlich durch die Lösung des entsprechenden linearen Gleichungssystems.

Abbildung von geschlossenen, einfachen Netzen auf die Kugel

Zur Einbettung von polygonalen Netzen mit Geschlecht 0 auf der Kugel bieten sich verschiedene Verfahren an. Ist die Geometrie sternförmig, so gibt es mindestens einen Punkt im Inneren, so dass die Abbildung der Knoten von diesem Punkt auf die Oberfläche einer Kugel um diesen Punkt bijektiv ist [Kent et al. 1991; Kent et al. 1992]. Das Problem ist damit gelöst.

Allerdings sind die meisten Formen nicht sternförmig. Dann bieten sich Relaxationsverfahren an, die an die Methoden im ebenen Fall angelehnt sind: Jeder Knoten wird im Zentrum seiner Nachbarn angeordnet, wobei allerdings die Knotenposition durch die Kugeloberfläche beschränkt sind. Diese quadratische Nebenbedingung kann man gut mit Hilfe von iterativen Verfahren erfüllen [Alexa 1999; Alexa 2000].

Zerlegung der Netze in einen gemeinsamen Teilgraphen

Zur Zerlegung von zwei polygonalen Netzen in einen gemeinsamen Untergaph $|L|$ geht man typischerweise davon aus, die Topologie L dieser Strukur sei gegeben oder wird vom Benutzer interaktiv spezifiziert. In jedem Fall muss der Benutzer die Positionen der Knoten in L auf den geometrischen Instanzen V_0, V_1 festlegen. Das Problem besteht darin, die Oberflächen von $\mathcal{M}_0, \mathcal{M}_1$ in Gebiete zu zerlegen, die mit den Facetten in L korrespondieren. Anders gesagt gilt es, die Kanten in L auf $\mathcal{M}_0$ bzw. $\mathcal{M}_1$ zu definieren, so dass die Topologie von L erhalten bleibt. Ein erster Ansatz ist die Verwendung von kürzesten Pfaden in den Graphen K_0 bzw. K_1 [Gregory et al. 1998; Bao & Peng 1998; Zöckler et al. 2000; Kanai et al. 2000]. Dies kann jedoch zu Problemen führen, weil die kürzesten Pfade im Graphen nicht notwendigerweise disjunkt sind. Vielmehr muss man bei der Bestimmung von Kanten Nebenbedingen beachten, so dass eine topologisch korrekte Zerlegung sicher gestellt ist [Praun et al. 2001].

Konstruktion eines Repräsentations-Netzes

Seien die Einbettungen W_0, W_1 der Netze $(V_0, K_0), (V_1, K_1)$ auf einem gemeinsamen Parameterraum D gegeben. Zu bestimmen ist eine Topologie K mit zwei Geometrien $V(0), V(1)$, so dass die ursprünglichen Formen (möglichst) exakt reproduziert werden:

$$\phi_V(0)(|K|) \approx \phi_{V_0}(|K_0|), \phi_V(1)(|K|) \approx \phi_{V_1}(|K_1|).$$

Das Problem ist dabei nicht die Bestimmung der Koordinaten $V(0), V(1)$, da sich diese aus den baryzentrischen Koordinaten direkt ergeben. Eventuell ist es sinnvoll, dabei nicht die Fläche des Netzes sondern eine glatte Approximation dieser Fläche zu verwenden. Die wesentliche Schwierigkeit ist die Bestimmung einer geeigneten Topologie K.

Es gibt zwei grundsätzlich verschiedene Wege zur Bestimmung von K: Entweder K enthält K_0 und K_1 und ergibt sich durch Überlagerung der Einbettungen (dem sogenannten Verschneiden) oder es wird ausgehend von der gemeinsamen Approximation L durch rekursive Verfeinerung ein genügend feines Netz konstruiert. Bei der Überlagerung von Netzen muss man ferner die Fälle von ebenen Teilnetzen und geschlossenen mannigfaltigen Netzen unterscheiden.

Verschneiden von polygonalen Netzen

Beim Verschneiden der Netze entstehen in jedem Schnittpunkt von zwei Kanten aus den beiden Netzen neue Knoten. Auf diese Weise entsteht das gesuchte gemeinsame Netz K.

Zum Verschneiden von planaren Graphen gibt es eine Reihe von Arbeiten [de Berg et al. 1997; Finke & Hinrichs 1995], die auch die asymptotische Schranke

für den besonderen Fall zweier geschlossener polygonaler Netze von $O(n+k)$ erreichen. Diese Techniken lassen sich allerdings nur schlecht auf unberandete Mannigfaltigkeiten anwenden.

Die grundsätzliche Idee zum Verschneiden der Netze besteht darin, beide Graphen zu traversieren und dabei die traversierten Kanten auf Schnitt mit den Facetten des anderen Netzes zu testen. Bei der Traversierung kann man dann die Kohärenz zwischen den Flächen ausnutzen [Alexa 2000].

Rekursive Verfeinerung

Die Strategie der rekursiven Verfeinerung startet mit dem gemeinsamen Teilgraph L. Dieser wird durch Einfügen von Knoten auf den Kanten und/oder Facetten nach bestimmten Regeln verfeinert. In der Ebene können die Einfügepositionen mittels baryzentrischer Koordinaten angegeben werden. Diese Koordinate erlaubt es, den Knoten mittels der bekannten Abbildung ϕ_V für beide Netze eine Position in $\mathbb{R}^3$ zuzuordnen [Lee et al. 1998].

Damit ensteht eine Kaskade von Netztopologien $L = L_0, L_1, L_2, \ldots$ wobei jedem Knoten die Position den Ursprungsgeometrien zugeordnet sind. Wegen des Erzeugungsprozesses entsteht zwangsläufig eine gemeinsame Netztopologie, die zudem nicht einmal gespeichert werden muss, da sie durch den Erzeugungsprozess bereits definiert wird. Da die Verfeinerung allerdings unabhängig von der ursprünglichen Gestalt ist, können scharfe Kanten unter Umständen nur unzureichend approximiert werden.

Bestimmung von Knoten-Pfaden

Ausgehend von der gemeinsamen Topologie K und den geometrischen Instanzen $V(0)$, $V(1)$ gilt es, für jedes $t \in]0,1[$ ein $V(t)$ zu finden, so dass die Knoten möglichst "schön" von $V(0)$ nach $V(1)$ interpoliert werden. Intuitiv wird man zumindest fordern, dass die Knotenpfade im Parameter T stetig sind und zwar auch in den Stellen 0 und 1 oder besser die Ableitung existiert und endlich ist: $\frac{dV}{dt} < \infty, t \in [0,1]$.

Der einfachste Ansatz ist lineare Interpolation der Knotenposition

$$V(t) = (1-t)V(0) + tV(1).$$

Falls sich die Ausgangsformen ähnlich sind, führt dies häufig zu befriedigen Ergebnissen. Ist aber die Orientierung der Objekte unterschiedlich, so sollte diese mit t angepasst werden [Cohen-Or & Carmel 1998; Cohen-Or et al. 1998; Alexa 2000]. Dazu kann man dem Übergang von $V(0)$ zu $V(1)$ eine affine Abbildung zuordnen, bspw. indem man unter allen Abbildungen diejenige Abbildung A sucht, die das Fehlerquadrat von $AV(0) - V(1)$ minimiert. Um nun $A(t)$ zu bestimmen, wird A mittels der Polarzerlegung in eine speziell orthogonale und eine symmetrische Matrix faktorisiert. Die Rotation wird dann über den Rotationswinkel interpoliert und

die symmetrische Matrix mit der Einheitsmatrix konvex kombiniert [Shoemake & Duff 1992; Alexa et al. 2000].

Meistens ist die Geometrieveränderung allerdings nicht ausreichend durch eine Transformation beschrieben. Dann kommt es zu Verformungen der Objektgeometrie während des Übergangs. Um solche Verformungen zu vermeiden, gibt es verschiedene Ansätze. Polygone kann man bspw. durch die Kantenlängen und Innenwinkel darstellen und diese Parameter interpolieren [Sederberg et al. 1993]. Allerdings ist dieser Ansatz schlecht auf polygonale Netze zu erweitern.

Ein besserer Ansatz ist, den durch die Oberflächenbeschreibung eingeschlossenen Raum in Simplizes zu zerlegen [Floater & Gotsman 1999; Alexa et al. 2000; Gotsman & Surazhsky 2001]. Da die Formen die gleiche Topologie K auf der Berandung haben, ist es möglich, auch das Innere so zu zerlegen, dass eine Isomorphie zwischen den Zerlegungen besteht. Mithin korrespondieren je zwei Simplizes in den Zerlegungen. Für jedes Paar kann man nun die ideale Transformation wie oben beschrieben bestimmen. Durch die gemeinsamen Knoten der Simplizes sind die idealen Pfade jedoch widersprüchlich. Die Knotenpfade entstehen daher durch Minimierung. Dabei wird die Abweichung der idealen affinen Transformationen der Simplizes von der des möglichen Pfades minimiert [Alexa et al. 2000]. Dieses Problem führt auf die einmalige Lösung eines schwach besetzten linearen Gleichungsystems sowie einer Matrix-Vektor Multiplikation für jeden Zeitschritt.

Bisher wurde nur die Interpolation aller Knoten in gleicher Weise von Ursprungs- zu Zielposition diskutiert. In vielen Fällen ist es aber auch interessant, Bereiche der Formen verschiedene Parameterwerte zwischen 0 und 1 zuzuordnen. Dies kann bei Interpolationsverfahren in absoluten Koordinaten zu Problemen mit der Stetigkeit führen. Daher ist es ratsam, solche zeitlich und örtlich variablen Interpolation in differentiellen Koordinaten zu beschreiben. Eine einfache Form differentieller Koordinaten für polyonale Netze sind Laplace-Koordinaten [Alexa 2001a]. Dabei wird jede Knotenposition relativ zu dem Schwerpunkt der unmittelbaren Nachbarn im Graph beschrieben. Die Rückwärtsabbildung von Laplace-Koordianten zu absoluten (euklidischen) Koordinaten führt auf ein lineares Gleichungssystem in den Knotenpositionen.

Räume aus Morphing-Verfahren

Zunächst wird der klassische Begriff Morphing abstrakt definiert und darauf aufbauend eine Definition für den Raum angegeben, der sich durch wiederholte Anwendung von Morphing-Verfahren auf eine Menge von Basisgeometrien ergibt. Es kann gezeigt werden, dass der Morphing-Raum mathematisch gesehen ein Vektorraum ist, wenn das verwendete Morphing-Verfahren gewisse Voraussetzungen erfüllt. Mit Hilfe dieser Kenntnis können Algorithmen zur Synthese und Analyse von Objekten gefunden werden, die nachweisbar asymptotisch ideal sind. Aber auch wenn der Morphing-Raum kein Vektorraum ist, können Elemente synthetisiert und analysiert werden. Die Algorithmen sind dabei unabhängig von den ver-

wendeten Objekten.

Für den Fall von polygonalen Netzen ist insbesondere die Erzeugung einer einheitlichen Topologie K aller Basen entscheidend. Dazu eignen sich nicht alle oben diskutierten Verfahren. Besonders gut geeignet ist die Erzeugung eines Netzes durch rekursive Verfeinerung, weil hier die Komplexität des Netzes nicht von der Anzahl der Basen sondern von der Komplexität der Formen abhängt. Das Ergebnis dieses Schrittes ist die gemeinsame Netztopologie K sowie eine Reihe von Basisgeometrien V_i.

Zur Knoteninterpolation eignet sich inbesondere lineare Interpolation der absoluten Koordinaten, weil damit die Vektorraumeigenschaften des Raums trivial gegeben sind. Ein Element des Raumes kann dann einfach durch einen Vektor von Gewichten w_i identifiziert werden: $V(\mathbf{w}) = \sum w_i V_i$. Die Erweiterung des Ansatzes auf nicht-lineare Interpolationsverfahren führt zwangsläufig auf einen nichtlinearen Raum.

Anwendungen in der Informationsvisualisierung

Die Visualisierung von abstrakten Daten geschieht meist durch die Abbildung der Datenattribute auf visuelle Skalen. Typische Besipiele für solche Skalen sind Ort, Farbe, Größe und Form [Chernoff 1973; Pickett & Grinstein 1988]. Zur Darstellung von Multiparameterdaten werden gerne Glyphen verwendet, die eine Kombination solcher Skalen darstellen.

Die Idee, Räume von Formen zur Generierung von Glyphen zu verwenden, liegt daher auf der Hand. Jede Basisform kann so ausgewählt werden, dass sie einen bestimmten Datenfall gut repräsentiert. Die Kombination der Formen ergibt dann die Glyphen für jedes Datum [Alexa & Müller 1998b; Müller & Alexa 1998].

Diese einfache Abbildung von Daten auf visuelle Repräsentationen erlaubt auch ein einfaches aber effektives Verfahren der Benutzer-Interaktion: Der Anwender kann bestimmten, bekannten Datenwerten eine bestimmte visuelle Repräsentation zuordnen. Auf der Basis von wenigen solchen direkten Zuordnungen wird eine Abbildungen vom Datenraum in den Raum der visuellen Repräsentation definiert [Alexa & Müller 1999b]. Diese kann entweder möglichst starr sein und die definierten Relationen nur approximieren oder z.B. mit den in der Approximation gerne verwendeten radialen Basisfunktionen die Relationen exakt erfüllen.

Der Prozess der Zurodnung geschieht dabei interaktiv, d.h. nach jedem Schritt kann eine neue Visualisierung generiert werden und auf dieser Basis neue – die Abbildung feiner spezifizierende – Relationen definiert werden.

Verwendung für geometrische Animationen

Wie zuvor gehen wir davon aus, eine Netztopologie K und mehrere Koordinatenvektoren V_i seien vorgegeben. Eine Kurve $\mathbf{w}(t)$ durch den Raum kann dann als

Animationssequenz $V(t) = V(\mathbf{w}(t))$ verstanden werden. Eine solche Darstellung einer Animation hat zwei wesentlich Vorteile:

1. Die Darstellung ist kompakt, vorausgesetzt die Anzahl der Basisvektoren ist kleiner als die Anzahl der Elemente der Animation.

2. Die Darstellung hat Semantik, da einzelne Elemente aus den Basisformen zusammengesetzt werden. Das erlaubt die Interpretation der Animationsteile sowie einen Austausch von charakteristischen Merkmalen.

Um eine solche Darstellung zu erzeugen, sind zwei Wege denkbar: Entweder ist der Raum vorgegeben und die Animation wird innerhalb des Raumes modelliert [Parke 1979; Müller et al. 2000; Alexa et al. 2001], oder die Animation ist vorgegeben und es gilt einen geeigneten Raum, sprich die Basisvektoren zu finden [Alexa & Müller 2000]. Der erste Weg kann z.B. eingesetzt werden, um animierte Avatare zu generiereren. Mit der folgenden Methode kann die Generierung von Basen aus Keyframe Animationen erreicht werden.

Die grundsätzliche Idee besteht darin, die Keyframes einer Animation als Basis eines linearen Raumes zu verstehen. Durch eine Basistransformation soll eine Basis gefunden werden, in der die Basen nach ihrer Wichtigkeit sortiert sind, so dass die Anzahl der Basen mit der Qualität der Animationsrepräsentation skaliert werden kann. Als erster Schritt wird die Animation auf affine Transformationen untersucht. Dabei wird jedem Keyframe eine affine Transformation zugeordnet, die die Hauptachsen der Kovarianzen mit dem kanonischen Koordinatensystem in Übereinstimmung bringt.

Die eigentliche Transformation basiert auf folgender Idee: Die wichtigste Geometrie ist der Durchschnitt aller Keyframe Geometrien. Alle Keyframes werden dann als Differenz zu dieser Durchschnittsgeometrie dargestellt. Die Durchschnittsgeometrie ist der erste Basisvektor des neuen Darstellungsraumes. Die Differenzen zur Durchschnittsgeometrie werden wiederum gemittelt, das Ergebnis ist die zweite Basis, usw.

Diese Transformation lässt sich mathematisch noch etwas wirkungsvoller beschreiben und heisst Hauptkomponentenanalyse. Da die Koordinaten der Keyframes als Vektoren beschrieben sind, kann man die Hauptkomponentenanalyse durch die Singulärwertzerlegung berechnen.

Die Singulärwertzerlegung liefert eine alternative Basis zur Darstellung der Animation. Meist sind nur wenige Basen notwendig um eine befriedigende Wiedergabe einer gegebenen Animation zu ermöglichen. Dies erlaubt eine effektive progressive Kompression von Animation.

X

Rückblick und Ausblick

In der Arbeit werden Verfahren zur Erzeugung von Räumen geometrischer Formen vorgestellt. Dabei wurden zum Stand der Forschung die folgenden Beiträge geleistet:

Morphing-Verfahren für polygonale Netze

Korrespondenzabbildung Ein Verfahren zur Korrespondenzabbildung topologischer Kugeln unter besonderer Beachtung von benutzerspezifizierten Knotenkorrespondenzen wurde vorgestellt [Alexa 1999; Alexa 2000].

Schneiden polygonaler Netze Ein neuer Algorithmus zum Schneiden polygonaler Netze wurde beschrieben, der vorherige Resultate verallgemeinert und asymptotische optimale Laufzeit hat [Alexa 2002b].

Orts-Zeit Spezifikation Im Bild-Morphing sind schon lange Verfahren bekannt, die es erlauben, Teile der Bilder örtlich und zeitlich unabhängig zu überblenden. Hier werden entsprechende Verfahren für Mannigfaltigkeiten präsentiert. [Alexa 2001a; Alexa 2002a].

Knotenpfade Eine Methode zur Bestimmung von Knotenpfaden wurde vorgestellt, die zu einer minimalen Verformung der lokalen Objektgeometrie führt [Alexa et al. 2000].

Visualisierung von Multiparameterdaten Lineare Räume von Formen haben sich als geeigente Repräsentation von Glyphen zur Informationsvisualisierung erwiesen. Zusätzlich wurde neue Formen der Interaktion zur Spezifikation der Abbildungsvorschrift vorgestellt [Alexa & Müller 1998b; Alexa & Müller 1999b].

Animation Lineare Räume von polygonalen Netzen sind eine geeignete Basis zur Spezifikation, Darstellung, Modifikation und progressiver Kommunikation sowie Speicherung von Animationen [Alexa et al. 2000; Alexa & Müller 2000].

Offene Fragestellungen, die sich aus dieser Arbeit ergeben, sind insbesondere topologische Probleme: Eine gegebene Netztopologie schränkt die Klasse der möglichen Transformationen und damit die Reichhaltigkeit der Formen in einem Raum stark ein. Dieses Problem könnte durch andere Repräsentation der geometrischen Form oder eine flexible Netztopologie gelöst werden. Darüber hinaus lässt die Robustheit der geometrischen Algorithmen z.T. noch stark zu wünschen übrig.

Acknowledgments

Many people have helped in one one way or another to make this dissertation happen. I would like to thank everybody who supported me.

My special thanks go to my advisor José L. Encarnação, for giving me the freedom to explore this interesting subject while helping me to stay on the right track. I thank Markus Gross for accepting to be on my thesis committee. I also owe a lot to Daniel Cohen-Or. His idea to invite me to Israel has initiated a series of visits, each of which has been a remarkable experience.

Many of the people I have been working with have helped me. In particular, my project leader Wolfgang Müller has been a source for countless valuable discussions. I am also grateful for his way of leadership, which has become an exemplar for me. I would also like to thank my current and former colleagues Johannes Behr, Uwe Berner, Erik Blechschmidt, Norbert Braun, Carola Eichel, Peter Frisch, Manfred Gaida, Paul Grimm, Ido Iurgel, Kai Kreuzer, Detlef Krömker, Thomas Rieger, Silke Romero, Michael Schneider, Ulrike Spierling, Francesca Taponecco, and Marc Weber.

I have had contact to several experts in the field of morphing who have inspired me and gave useful advice. This work might not exist without Herbert Edelsbrunner who introduced me to the idea of a shape space. I thank Craig Gotsman, Reinhard Klein, David Levin, and Michela Spagnuolo for many fruitful discussions. Jed Lengyel has given me access to some of the animated sequences used in this work (The chicken character was created by Andrew Glassner, Tom McClure, Scott Benza, and Mark Van Langeveld. This short sequence of connectivity and vertex position data is distributed solely for the purpose of comparison of geometry compression techniques). I have reused material from other excellent work in mesh morphing and I thank Takashi Kanai, Ming Lin, and Peter Schröder for allowing me to do so.

Part of this work was done when I was visiting Tel-Aviv University supported by the Hermann Minkowski - Minerva Center for Geometry. Thanks to David Levin and Daniel Cohen-Or for hosting me and making my stay so enjoyable.

I am deeply indebted to my parents. Where I am today is no small part due to their encouragement and support. I would like thank my friends for their understanding that this work took much of my time. And I am grateful for the time I shared with Regina.

Contents

Chapter 1

Introduction

Art always serves beauty,
and beauty is the joy of possessing form,
and form is the key to organic life
since no living thing can exist without it.
Boris Pasternak, Doctor Zhivago

Every thing has a shape. We can see, touch, even hear shape. It is the fundamental concept for interaction with the world we live in. We compare one shape with another to assess it, to put it into context: Every shape can be seen as as a variation of another shape or the combination of a few shapes.

A formalism is needed to communicate shape. Many such formalisms exist, however, most of them describe shape in an absolute way, without relating to other objects. This cannot be avoided if only one single object has to be described. If many (relating) objects are concerned, an absolute description not only misses information about the relation to other objects it might also be inefficient since the differences of shapes could be described more compactly than the whole shape.

This work mainly deals with the description of shapes as *compounds* of a set of *base shapes*. Each base shape is described absolutely as set of planar patches, being an approximation of the "real" shape. These base shapes can be combined and, altogether, form a *space of shapes*. Such representation of shape is particularly suited to deal with sets of shapes with some common ground. For example, think of a space of shapes representing faces: This space is more appropriate for modeling a face, generating or storing animated sequences of faces. Limiting the possible choices means, in this context, the need for less information or storage space to generate or represent the intended object. An illustration is given in Figure 1.1.

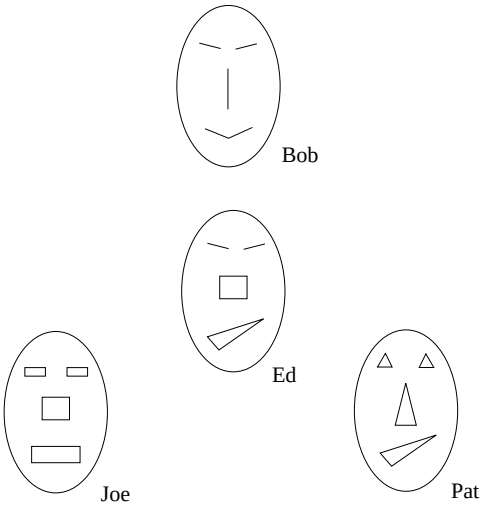


Figure 1.1: Describing shapes as a compound of base shapes. It seems natural to describe Ed as having the eyes of Bob, the nose of Joe, and the mouth of Pat rather than giving details of the shape of each feature. If we wanted to communicate information about Ed's face, the description in terms of Bob, Joe, and Pat is also much smaller and, yet, absolutely precise. Note that the faces of Bob, Joe, and Pat allow to generate a variety of other faces. This could be useful to model a face that can be described in terms of their features. This space of faces also contains sequences smoothly transforming from one face to another. Thus, it allows to generate and store animated sequences.

1.1 Technical motivation

In computer graphics, models of three-dimensional shapes are nowadays mainly represented as *meshes*. A mesh contains a set of vertices describing geometric positions (and other attributes such as color, etc.) and topological information describing edges containing vertices and forming faces. Meshes are universal in the sense that they can represent every shape with arbitrary precision (assuming infinite space to store the description).

In many applications one deals not only with one single mesh but with many meshes. The most prominent example are geometric animations, which is typically stored a set of meshes describing the shape over time. We like to exploit the idea of a shape space, where shapes are described as the combination of a few base shapes. Here, base shapes are meshes, and all combinations are meshes.

We start exploring this idea by looking at the simple case of only two base meshes. The main idea of this work is to use *morphing* techniques to generate the family of shapes described as the combination of two base shapes. Morphing techniques are used to generate smooth transitions from one object to another. They have become popular and widespread in the special effects industry but have appli-

cations in many areas such as medical imaging and scientific visualization. We can say that a morph represents the family of shapes generated by two base shapes, i.e. the space is one dimensional. By adding a third base shape and morphing between an element of the family resulting from the first two base shapes we add another dimension. This process can be repeated to add any number of dimensions.

Such spaces of shapes allow to represent each shape in the space with a vector of scalars not longer than the number of base objects spanning the space. Assuming the number of base shapes is relatively small with respect to the amount of information needed to describe a single shape, this is an extremely compact and meaningful way of describing a shape.

Why is the representation meaningful? Imagine a set of faces (smiling, frowning, blinking, staring, etc.) comprising the base of a space. If we want to generate a particular expression we simply describe the face in terms of the features we want. The modeling process is intuitive and simple. In addition, if such a face has to be stored or communicated only the small vector is needed.

The major aim of this work is to build spaces of polyhedral objects and demonstrate their usefulness in practical applications. However, at the current state of science even morphing between two polyhedral objects is a difficult process. For that reason, a large part of this work is dedicated to generating morph sequences between two meshes. This is a remunerative subject in itself, as morphing object representations is superior to morphing representations of space (such as images).

1.2 Overview & Framework

The first part of this work is dedicated to mesh morphing techniques. Mesh morphing techniques involve computations on the geometry as well as the connectivity of meshes. For simplicity this report concentrates on triangle meshes. In the context of morphing it seems to be acceptable to triangulate polygonal meshes prior to the application of a morphing technique. To classify and understand mesh morphing techniques it is helpful to use the now widespread terminology from Spanier [1966]. A mesh $\mathcal{M}$ is described by a pair (K, V), where K is a simplicial complex representing the connectivity of vertices, edges, and faces and $V = (\mathbf{v_1}, \ldots, \mathbf{v_n})$ describes the geometric positions of the vertices in $\mathbb{R}^d$, where typically $d = 3$.

The abstract complex K describes vertices, edges, and faces as $\{0, 1, 2\}$-simplices, that is, edges are pairs $\{i, j\}$, and faces are triples $\{i, j, k\}$ of vertices. The *topological realization* maps K to a simplicial complex $|K|$ in $\mathbb{R}^n$: The vertices are identified with the canonical basis of $\mathbb{R}^n$ and each simplex $\mathbf{s} \in K$ is represented as the convex hull of the points $\{\mathbf{e_i}\} \in \mathbb{R}^n, i \in \mathbf{s}$ Thus, each 0-simplex is a point, each 1-simplex is a line segment, and each 2-simplex is a triangle in $\mathbb{R}^n$.

The *geometric realization* $\phi_V(|K|)$ is a linear map of the simplicial complex $|K|$ to $\mathbb{R}^d$, which is defined by associating the basis vectors $\mathbf{e_i} \in \mathbb{R}^n$ with the vertex positions $\mathbf{v_i} \in V$. The map ϕ_V is an *embedding* if ϕ_V is bijective. The importance of an embedding is that every point $\mathbf{p}$ on the mesh can be uniquely represented

with a barycentric coordinate $\mathbf{b}$, i.e. $\mathbf{p} = \phi_V(\mathbf{b})$. Such barycentric coordinates have at most three non-zero components and specify the position of a point relative to a simplex. If the point is coincident with a vertex it is a canonical basis vector, if the point lies on an edge it has two non-zero components, otherwise it has three and lies on a face.

The neighborhood ring of a vertex $\{i\}$ is the set of adjacent vertices $\mathcal{N}(i) = \{j | i, j \in K\}$ and its star is the set of incident simplices $\mathcal{S}(i) = \bigcup_{i \in s, s \in K} s$.

In the classical setting of mesh morphing two meshes $\mathcal{M}_0 = (K_0, V_0)$ and $\mathcal{M}_1 = (K_1, V_1)$ are given. The goal is to generate a family of meshes $\mathcal{M}(t) = (K, V(t)), t \in [0, 1]$ so that the shape represented by the new connectivity together with the geometries $V(0)$ and $V(1)$ is identical with the original shapes, i.e. $\phi_{V(0)}(|K|) = \phi_{V_0}(|K_0|)$ and $\phi_{V(1)}(|K|) = \phi_{V_1}(|K_1|)$. Most of the time the paths $V(t)$ are required to be smooth. The generation of this family of shapes is typically done in three subsequent steps:

1. Finding a correspondence between the meshes. More specifically, computing coordinates W_0, W_1 that lie on the other mesh, i.e. $W_0 \in \phi_{V_1}(|K_1|)$ and $W_1 \in \phi_{V_0}(|K_0|)$. Each coordinate in W_0, W_1 is represented as a barycentric coordinate with respect to a simplex in the other mesh. Note that ϕ_{W_0} will not map $|K_0|$ to $\phi_{V_1}(|K_1|)$ (and vice versa), as only the vertices are mapped to the other mesh but not the edges and faces. Particularly important is the alignment of automatically detected or user specified features of the meshes. The process of finding correspondence between meshes is discussed in Chapter 2.

2. Generating a new, consistent mesh connectivity K together with two geometric positions $V(0), V(1)$ for each vertex so that the shapes of the original meshes are reproduced. The traditional morphing approach to this problem is to create a superset of the simplicial complexes K_0 and K_1. However, remeshing techniques as used in multi-resolution techniques are also attractive. Methods to generate such representation are discussed in Chapter 3.

3. Creating paths $V(t), t \in]0, 1[$ for the vertices. While in general this is an aesthetic problem, several constraints seem reasonable to help in the design process. For example, in most applications the shape is not expected to collapse or self intersect and, generally, the paths are expected to be smooth. Techniques tackling the path problem are discussed in Chapter 4.

An illustration of this process is shown in Figure 1.2.

Using such mesh morphing techniques, a shape space can be build. However, it is interesting to know the mathematical properties of the resulting space from the mathematical properties of the morphing technique. This calls for a theoretical model of shape spaces. In addition, some mesh morphing techniques are better suited to be extended to more than one dimension. These issues are analyzed in Chapter 5.

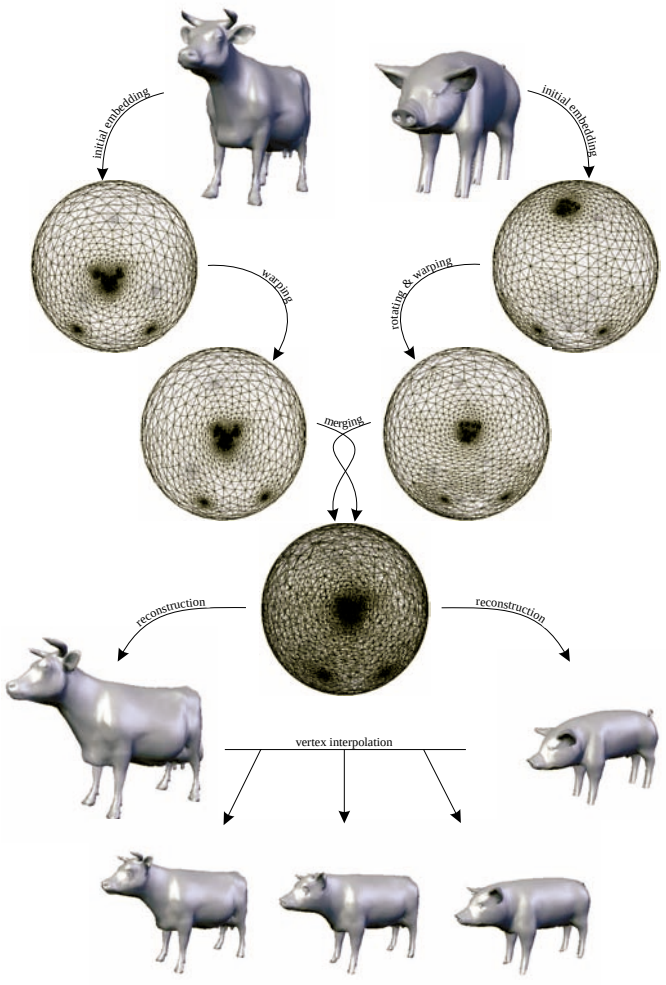


Figure 1.2: The process of mesh morphing illustrated at the example of meshes that allow a mapping to the sphere. First, the meshes are embedded on the sphere, thus, establishing correspondence between the set of vertex positions. The correspondence is refined using information about feature correspondence. The embeddings are used to generate one mesh containing, which can represent both input meshes. A vertex interpolation scheme yields the morph.

The last two chapters of this work show applications of this approach. On one hand, the idea of mapping from an abstract vector to a shape is used for information visualization. On the other hand, spaces of meshes are used to generate and compress geometric animations. This seems to be the most fruitful application of this work so far.

While in general shapes will be represented as meshes throughout this work, other types of representations (polygons,images) are used occasionally. In some cases an algorithm was developed for such other representations and we believe it is best explained for that representation and later extended to meshes. However, sometimes the extension has not been found, yet, we present the algorithm for the sake of completeness of this work.

1.3 Context of prior work

The idea of a *shape space* has been used occasionally for a long time. The term seems to be first used by Herbert Edelsbrunner (in a private communication) and has been published later [Cheng et al. 1998; Edelsbrunner 1999]. Spaces of shapes have been used for a long time in visual computing, particularly face recognition applications (Eigenfaces [Kirby & Sirovich 1990; Turk & Pentland 1991]). Recently, a linear space has been used to model faces [Blanz & Vetter 1999], however, the space is particularly designed for faces. In animation, a configuration space has been limited by forming cross-products of small linear spaces [Ngo et al. 2000]. It seems that most approaches to date use a space to solve a particular problem rather than viewing the space as a fundamental concept for shape representation (as in [Cheng et al. 1998; Edelsbrunner 1999] and this work).

Mesh morphing techniques have drawn much attention recently, following the success of image morphing techniques. Image morphing has been developed for special effects in feature films and commercials (see [Wolberg 1998] for a recent survey on image morphing). The extensions of some of the concepts in image morphing to explicit representations of shapes (such as meshes) are difficult because of topological restrictions. Images do not carry the topology of the objects they depict and, therefore, no problems from different topology can arise. The topology problem as well as feature alignment and local control over the morph are recent topics of research in mesh morphing. Lazarus & Verroust [1998] surveys 3D morphing algorithms. This work has also resulted in survey on mesh morphing [Alexa 2001b; Alexa 2002b].

1.4 Contributions

The primary contributions of this work to the field of computer graphics are:

Mesh morphing Several contributions to the field of morphing meshes have been made.

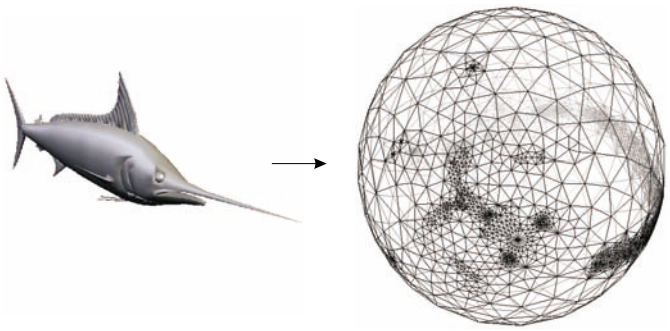
- **Feature alignment** A mesh morphing technique for topological spheres is introduced, which allows for particular easy feature specification [Alexa 1999; Alexa 2000].
- **Mesh merging** A new algorithm for generating one mesh from two input meshes (mesh overlay) is presented, generalizing previous results. In contrast to other approaches, the algorithm is asymptotically optimal [Alexa 2002b].
- **Local control** In image morphing one can easily specify which region transforms when in the morph, e.g., first the nose, then the eyes and then the rest of a face. We present similar techniques for mesh morphing [Alexa 2001a; Alexa 2002a].
- **Vertex path** A method to generate intuitive and well behaving vertex paths in morphing is introduced. The paradigm employed is that the object is transformed rigidly as much as possible to avoid unnecessary deformations [Alexa et al. 2000].

Visualization of multiparameter data Morph spaces are shown to be a useful way to generate glyphs and icons in the visualization of multiparamter data. Additionally, the paradigm allows for a flexible and ituitive generation of the visualization [Alexa & Müller 1998b; Alexa & Müller 1999b].

Geometric animations Spaces of meshes are shown to be a particularly elegant and effective way to generate, store, and communicate geometric animations. The resulting compression is progressive and the achieved compression ratios progress over previous work [Alexa et al. 2000; Alexa & Müller 2000].

Chapter 2

Correspondence of shapes



In this chapter we aim at finding corresponding vertex positions on two or more shapes. We assume the boundaries of the shapes to be manifolds and homeomorphic (as long as not noted otherwise). Given two manifold meshes $\mathcal{M}_0$ and $\mathcal{M}_1$, the result of this procedure is a set of barycentric coordinates B_0 so that the geometry $W_0 = \phi_{V_1}(B_0)$ of the barycentric coordinates on $\mathcal{M}_1$ is an embedding ϕ_{W_0} of $\mathcal{M}_0$ on the surface of $\mathcal{M}_1$, and vice versa. The idea is that this mapping of vertices from one mesh to the other accomplishes the main part of a bijective mapping between the surfaces of $\mathcal{M}_0$ and $\mathcal{M}_1$. After this step only the edges and faces have to be adjusted accordingly.

The process is typically done by finding a common parameter domain D for the surfaces. By mapping each surface bijectively to that parameter domain, the mapping between the shapes is established. The typical parameter domains for meshes in the context of morphing are the sphere $\mathbb{S}^2$ (in case the meshes are topological spheres) or a collection of topological disks represented as a piecewise linear parameter domain L. In case of the disks, the meshes have to be decomposed into isomorphic structures of disks (which requires them to be homeomorphic). A major constraint is to take into account user specified or automatically generated feature correspondences (i.e. vertex-vertex correspondences). Depending on the approach chosen, this is done by re-parameterization or by decomposing the

meshes according to the feature correspondence.

In case of mapping to a sphere, an embedding ϕ_S with $S = \{s_0, s_1, \ldots\}, s_i \in \mathbb{R}^3, |s_i| = 1$ is computed. The embeddings on the sphere are aligned according to the feature correspondence using a bijective map f that maps spheres into spheres.

$$\begin{array}{ccc} \{i\} \in K_0 & \xrightarrow{W_0} & \phi_{V_1}(B_0) \\ {\scriptstyle \phi_{S_0}}\downarrow & & \uparrow{\scriptstyle \phi_{S_1}^{-1}} \\ \mathbb{S}^2 & \xrightarrow[f]{} & \mathbb{S}^2 \end{array}$$

The main problems in this approach are to compute the vertex coordinates S_0, S_1 on the sphere and the re-parameterization f.

The decomposition approach is more general and more difficult. In addition to generate embeddings of the topological disks one has to decompose the meshes in an isomorphic way, taking possible feature correspondences into account. Formally, an abstract simplicial complex L consisting of a subset of the vertices in K_0, K_1 is used as coarse approximation of both meshes:

$$\phi_{V_0}(|L|) \approx \phi_{V_0}(|K_0|), \phi_{V_1}(|L|) \approx \phi_{V_1}(|K_1|)$$

Typically, L is topological minor of K_0 as well as K_1, i.e. it is a partition of the meshes. Vertices in K_0, K_1 are identified with a face in L and all vertices belonging to a particular face are embedded in its planar shape. Thus, the common parameter domain is the topological realization $|L|$, where each vertex is represented with a barycentric coordinate with respect to a particular face in L. This requires to embed pieces of the mesh in the plane.

$$\begin{array}{ccc} \{i\} \in K_0 & \xrightarrow{W_0} & \phi_{V_1}(B_0) \\ {\scriptstyle \phi_{L_0}}\downarrow & & \uparrow{\scriptstyle \phi_{L_1}^{-1}} \\ |L| & \xrightarrow[f]{} & |L| \end{array}$$

Following, techniques to embed simply-connected bounded and unbounded meshes in the plane and on the sphere are explained. Then, approaches to dissect the meshes into isomorphic patch-networks (or, equivalently, inducing base-domains $|L|$ on $\mathcal{M}_0, \mathcal{M}_1$) are discussed. After these basic embedding steps re-parameterization for feature alignment is introduced. Finally, some comments on rarely mentioned details in the correspondence problem are given.

2.1 Parameterizing topological disks

Simply-connected parts of the boundary of three dimensional shapes are homeomorphic to a disk and, therefore, called topological disks. In order to find a parameterization of such pieces we need a bijective map of a bounded, simply connected mesh to the plane.

In our application we need to find a bijective map between patches. Thus, it is necessary to constrain the boundary of the patches to a particular shape. Here, we concentrate on mapping an arbitrary bounded and simply connected mesh to a unit disk so that boundary vertices of the mesh lie on the unit circle. This limits the applicability of several parameterization approaches, which allow the boundary of a triangulated surface to be non-convex or not to be fixed a priori to achieve smoother or less distorted mappings [Hormann & Greiner 2000; Lévy & Mallet 1998; Zigelmann et al. 2002].

In a first step the boundary vertices are fixed on the unit circle. First, the three vertices from the base domain L are fixed in an equiangular way. This is necessary to make sure that adjacent faces in the base domain have a continuous parameterization across base domain edges. The remaining boundary vertices are fixed so that the arc lengths between neighboring vertices are proportional to the original edge lengths. The remaining (interior) vertices are free and their position is determined by a relation to neighboring vertices.

Most of the publicized approaches to solve this task minimize a quadratic error functional expressed as the vertex position relative to its neighbors. This boils down to solving a system of linear equations. Non-linear approaches either use higher order functionals to be minimized [Hormann & Greiner 2000; Lévy & Mallet 1998] or are of algorithmic nature (e.g. Gregory et al. [1999], which is discussed after the linear techniques).

More specifically, let $\{\mathbf{v_i}\}$ be the vertices to be mapped to the disk so that the free interior vertices have indices $0 \leq i < n$ and the fixed boundary vertices have indices $n \leq i < N$. We aim at finding positions $\mathbf{w_i}$ in the plane with $|\mathbf{w_i}| = 1, n \leq i < N$. The mapping is bijective if and only if no edges cross or, alternatively,

$$\forall (i,j,k) \in K. \det \begin{pmatrix} \mathbf{w_{i_x}} & \mathbf{w_{i_y}} & 1 \\ \mathbf{w_{j_x}} & \mathbf{w_{j_y}} & 1 \\ \mathbf{w_{k_x}} & \mathbf{w_{k_y}} & 1 \end{pmatrix} > 0. \tag{2.1}$$

However, this quadratic expression is awkward to use as a criterion to guarantee that the planar embedding is valid, which is why most approaches resort to the more restrictive but sufficient linear conditions.

In the following we discuss three ways to define a linear system, whose solution yields positions for the vertices. In addition, the Divide&Conquer approach of Gregory et al. [1998, 1999] is explained.

Barycentric mapping

Tutte [1963] has shown how to embed planar graphs in the plane using a barycentric mapping. In our restricted setting, the idea is simply to place every interior vertex at the centroid of its neighbors:

$$\mathbf{w_i} = \sum_{j \in \mathcal{N}(i)} \frac{1}{|\mathcal{N}(i)|} \mathbf{w_j} \tag{2.2}$$

Setting $\Lambda = \{\lambda_{i,j}\}$ with

$$\lambda_{i,j} = \begin{cases} d_i^{-1} & \{i,j\} \in K \\ 0 & \{i,j\} \notin K \end{cases} \tag{2.3}$$

this can be written as the mentioned system of linear equations

$$(I - \Lambda) \begin{pmatrix} \mathbf{w_0} \\ \mathbf{w_1} \\ \cdots \\ \mathbf{w_{n-1}} \end{pmatrix} = \begin{pmatrix} \sum_{i=n}^{N-1} \lambda_{0,i} \mathbf{w_i} \\ \sum_{i=n}^{N-1} \lambda_{1,i} \mathbf{w_i} \\ \cdots \\ \sum_{i=n}^{N-1} \lambda_{n-1,i} \mathbf{w_i} \end{pmatrix} \tag{2.4}$$

The matrix $(I - \Lambda)$ has full rank and, thus, there is exactly one solution. Tutte [1963] has shown that this solution is a valid embedding of the mesh.

Note, that the shape of the mesh has no effect on the placement of vertices in the plane. All information for the embedding comes from K and it is clear that the embedding cannot reflect geometric properties contained in V of the mesh. In the following we try to incorporate information about the original shape.

Shape preserving parameterization

In the barycentric mapping the weights λ contain only topological information. Floater [1997] determines weights that reflect the local shape of the mesh. More precisely, the λ are so chosen that the angles and lengths of edges around a vertex are taken into account.

To compute the weights for a particular vertex $\mathbf{v_i}$ this vertex is placed in the origin and incident edges are laid out in the plane using the original edge lengths and angles proportional to the original angles. This is assumed to be the ideal parameterization $\mathbf{w'_i}$ of the mesh with respect to $\mathbf{v_i}$.

The weights are computed in way that would result in placing $\mathbf{w_i}$ in the origin if the neighbors $\mathbf{w'_j}$ were fixed and the system of equations had to be solved. Thus, we have

$$\mathbf{w_i} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \sum_{j \in \mathcal{N}(i)} \lambda_{i,j} \mathbf{w'_j} \tag{2.5}$$

and

$$1 = \sum_{j \in \mathcal{N}(i)} \lambda_{i,j}. \tag{2.6}$$

If $\mathbf{v_i}$ has only three neighbors this exactly determines the positive weights, for more than three neighbors a positive solution has to be chosen from the space of possible solutions. Note that positivity results in convex combinations, which are necessary to assure a valid embedding. Floater presents a method to compute reasonable weights, which are guaranteed to be positive: Take the cyclically ordered set of neighbors $j_k \in \mathcal{N}(i), k \in \mathbb{Z}_{|\mathcal{N}(i)|}$. Determine sets of weights $\lambda_{i,j}(k)$ with respect

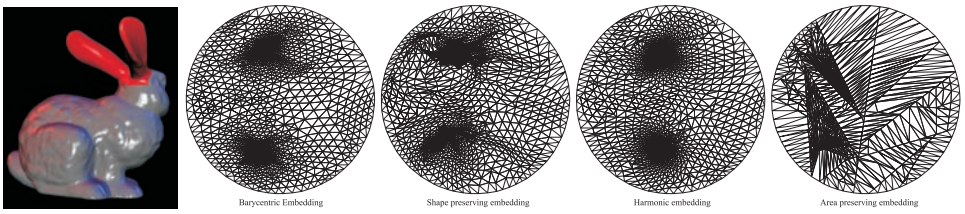


Figure 2.1: A part of a mesh parameterized on the unit disk using different mapping techniques. The original geometry is highlighted in red. A barycentric mapping (see Section 2.1) does not reflect the geometry of the mesh. The shape preserving embedding tries to capture the local shape of the mesh by locally approximating conformal maps (see Section 2.1). Discretized harmonic embeddings minimize metric distortion (see Section 2.1). The area preserving embedding is a recursive process, which aims at approximating the original area of triangles (see Section 2.1).

to three subsequent neighbors j_k, j_{k+1}, j_{k+2}. This yields non-negative $\lambda_{i,j}(k)$ for each k. These weights are averaged to yield the final weights:

$$\lambda_{i,j} = \frac{1}{|\mathcal{N}(i)|} \sum_k \lambda_{i,j}(k) \tag{2.7}$$

The positions w_i are computed by solving (2.4). Recently, Floater [2002, 2001] has proven a generalization of Tutte's theorem, which shows that it is sufficient that each vertex is a convex combination of its neighbors.

Discrete harmonic mappings

Harmonic mappings are a concept found in several fields in mathematics using differentials. Harmonic maps are often described as the function u among all functions mapping to a given domain Ω that minimize the Dirichlet energy

$$E_D(u) = \frac{1}{2} \int_\Omega |\nabla u|^2. \tag{2.8}$$

Pinkall and Polthier [1993] show how to discretize this problem for triangles, so that weights are derived per vertex and neighbor leading to a system of linear equations of the form of Eq. (2.4). A somewhat clearer derivation can be found in a more recent work of Polthier [2000]. There, it is shown that the discrete Dirichlet energy is

$$E_D(u) = \frac{1}{4} \sum_{i,j|\{i,j\}\in K} (\cot \alpha_{i,j} + \cot \beta_{i,j}) |\mathbf{v_i} - \mathbf{v_j}|^2,$$
$$\alpha_{i,j} = \angle(i, k_0, j), \beta_{i,j} = \angle(i, k_1, j), \{i, j, k_c\} \in K \tag{2.9}$$

and that the minimizer $\nabla E = 0$ solves

$$0 = \frac{1}{2} \sum_{j \in \mathcal{N}(i)} (\cot \alpha_{i,j} + \cot \beta_{i,j})(\mathbf{v_i} - \mathbf{v_j}) \tag{2.10}$$

at each vertex i. This leads to weights

$$\lambda_{i,j} = \begin{cases} \frac{\cot \alpha_{i,j} + \cot \beta_{i,j}}{\sum_{j \in \mathcal{N}(i)} (\cot \alpha_{i,j} + \cot \beta_{i,j})} & \{i,j\} \in K \\ 0 & \{i,j\} \notin K \end{cases} \tag{2.11}$$

which are used to obtain an embedding by solving Eq. (2.4).

Another formulation, which is probably better known in the graphics community, is given by Eck et al. [1995].

Area preserving Divide&Conquer mapping

Gregory et al. [1998, 1999] describe a recursive algorithm that aims at preserving the area of triangles in the mapping. The idea is to induce the mapping be recursively dividing the patch into two pieces, which are then mapped independently. The dissections are so chosen that the ratio of areas in the original mesh and the embedding are approximately the same.

In particular, two diametrical vertices on the boundary of a patch are chosen. A shortest path is computed using Dijkstra's algorithm. This path is mapped to the segment connecting the two vertices. Then the path is altered to minimize the difference of the ratio of areas in the embedding and on the triangulated surface. The segment divides the patch into two halves, which are treated in the same way, until all vertices are mapped.

Comparison and Conclusion

We have embedded parts of a mesh using the four approaches presented above. Note that the solution of matrix equation (2.4) could be performed using hierarchical techniques [Lee et al. 1998; Hormann et al. 1999], which is equivalent to using multi-grid methods. However, the matrix has sparse structure and we have found it sufficient to use iterative solvers exploiting the sparseness.

Some of the results of the comparison are shown in Figure 2.1. It is apparent that the general structure of larger and smaller triangles is very similar in all embeddings generated using linear optimization techniques. This suggests that connectivity is the major factor in these type of embeddings. Changing the weights used to compute the embedding only changes the local behavior of the embedding. They share the problem of area compression: Inner triangles have much less area than outer triangles. The area preserving scheme eliminates this problem at the cost of distorted triangle shapes.

In fact, all these parameterizations might be unusable due to the high ratio of either areas or angles and the limited precision of floating point numbers. It

has been observed that the base domain should have enough “skin” to allow for a reasonable parameterization of the mesh.

The small differences in local shape do not seem to have much influence on the resulting correspondence of the shapes. This is even more true when local features of the shapes are aligned by re-parameterization (see Section 2.4).

2.2 Parameterizing topological spheres

Unbounded simply-connected 2-manifolds are called topological spheres because they are homeomorphic to spheres. A natural parameter domain for such shapes is, therefore, a unit sphere.

2.2.1 Star shapes

Kent et al. [1991, 1992] were the first to present techniques to map certain classes of genus 0 meshes to a sphere. A particularly simple class of objects are convex shapes. A convex shape has the property that a straight line connecting any two boundary points of the shape lies completely inside the model. Thus, all points are visible form any interior point of the shape and a projection through an interior point onto an enclosing sphere is necessarily bijective.

A generalization of this idea extends the class of shapes to star shapes. Such shapes have at least one interior point so that straight lines connecting this interior point with boundary points lie completely inside the shape. Interior points with this property are called star points. Obviously, projecting the boundary points of a shape through a star point onto an enclosing sphere is a bijective mapping. Specifically, if point O is visible from all vertices of the mesh then translate all points so that O coincides with the origin. Then normalize all vertex coordinates. These vertex coordinates are the parameterization of the mesh vertices on a unit sphere. An illustration is given in Figure 2.2

The only problem is to determine whether a shape is star shaped and if so to find a star point. For piecewise linear shapes (meshes) this can be done by intersecting half-spaces induced by the face elements of the mesh. The intersection of all half-spaces is called kernel. If the kernel is non-empty the mesh is star shaped and every point inside the (convex) kernel is a suitable star point. The kernel of a mesh in 3D can be computed in $O(n \log n)$ using standard techniques [Preparata & Shamos 1985].

2.2.2 Star shapes around an axis

However, genus 0 objects can be parameterized not only on the sphere. Lazarus and Verroust [1997] parameterize the polyhedra over an axis and two spherical caps at the ends of the axis. The objects which are suitable for this approach can be considered as star-shaped around a 3d curve. The curves have to be specified by the user for each input object. The curves are used as axes to parameterize the object

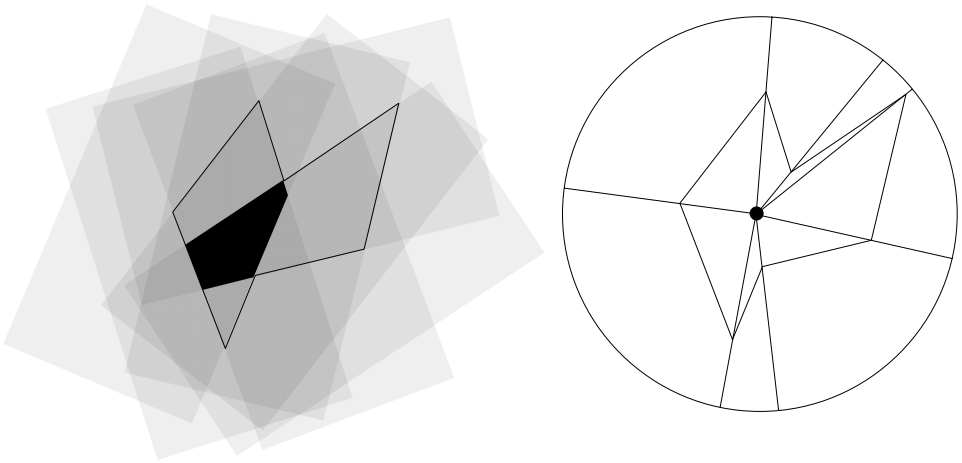


Figure 2.2: A polygonal star shape and its projection to a circle. The kernel of a star shape is the intersection of all open half spaces over the edges (faces in case of a polyhedron). Every point in the kernel induces a bijective mapping to the circle by projection.

onto three sheets. One domain results from cross-sections which are orthogonal to the axis, and two result from hemispherical regions around the endpoints of the axis.

2.2.3 Curve evolution

Carmel and Cohen-Or [1998] use curve evolution to find a parameterization of a mesh. The curve evolution process iteratively deforms the curve over time. The deformation of the curve from one time instance to the next is defined by curvature and normal field of the curve. Points of the curve are moved along normal directions with a step width depending on the local curvature. Using the right parameters, curve evolution will transform a simple curve into a convex curve so that the curve is simple in all time steps. Carmel and Cohen-Or adapt curve evolution for polygons and polyhedra since the theory for the continuous case does not hold in the discrete one. See Figure 2.3 for an illustration.

2.2.4 Simplification

Shapiro and Tal [1998] seem to be the first to present a reliable scheme that turns arbitrary genus 0 polyhedra into convex shapes. They first simplify the shape using vertex removal until the simplified shape is a tetrahedron. Only vertices with valence 3,4, and 5 are removed. Since the mesh is triangular such vertices always exist: It follows easily from the Euler-Poincare formulas that the average degree in any triangular (surface) mesh is less than 6. Thus, at least one vertex with degree

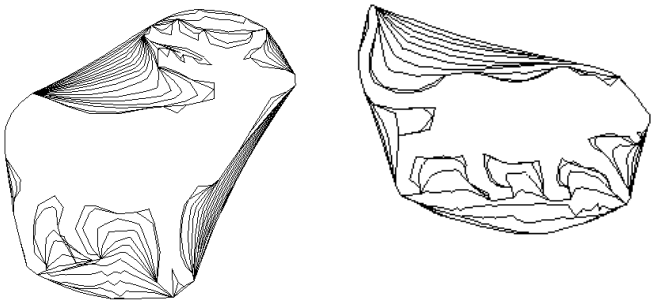


Figure 2.3: Two examples for curve evolution adapted to polygons.

strictly less than 6 has to exist.

Once the shape is simplified to a tetrahedron, vertices are re-attach making sure that the shape stays convex. More specifically, it is shown how to attach vertices with degree 3, 4 and 5 to a convex shape so that the shape stays convex. More specifically, if a vertex $\{i\}$ has to be added to a face f, its position has to be outside the convex hull of the current mesh but inside the kernel of faces adjacent to f.

2.2.5 Spring embedding

We have introduced a variation of spring embedding to embed polyhedra on the unit sphere [Alexa 1999; Alexa 2000]. In spring embedding algorithms, one tries to minimize a potential defined as

$$W = \sum_{\{i,j\}\in E} \kappa_{i,j} \left\| \mathbf{w_i} - \mathbf{w_j} \right\|^2 . \tag{2.12}$$

Algorithms based on this paradigm produce nice results in the plane (see the previous section). In the planar case, the embedding is supported by a fixed peripheral cycle of the graph. Obviously, a peripheral cycle does not make sense on a closed manifold. But, if none of the vertices is fixed, the minimum energy state is reached when all vertices coincide.

Our solution to this problem is as follows: Start with a reasonably equal distribution of the vertices on the sphere. During the relaxation towards a minimum energy state longer edges are penalized. Because the collapse of the vertices into one point has to pull at least one triangle over the equator, penalizing long edges effectively prevents the vertices from collapsing.

More precisely, in each step of the relaxation process, vertex i is moved

$$\mathbf{p_i} = c \frac{\sum_{\{i,j\}\in E} (\mathbf{w_i} - \mathbf{w_j}) \left\| \mathbf{w_i} - \mathbf{w_j} \right\|}{|\mathcal{N}(i)|} \tag{2.13}$$

Multiplying $(\mathbf{w_i} - \mathbf{w_j})$ by its length results in a quadratic weight for the edge lengths, such that longer edges are shortened. The constant c is used to control

the overall move length. With $c = 1$, the relaxation runs robustly but not very efficiently. For very short edges, the quadratic weight makes moves very short and, thus, convergence very slow. Therefore, it is advisable to scale up the move length proportionally to the inverse of the longest edge incident upon one vertex. The result is a much faster convergence.

Because $(\mathbf{w_i} + \mathbf{p_i})$ is not necessarily on the unit-sphere, we normalize the term.

Gumhold [2000] has reported another solution to the same problem, which is simpler and more elegant: The sphere is re-centered after each relaxation round, i.e.

$$\mathbf{w_i} = \frac{1}{n}\left(\sum_{j \in K} \mathbf{w_j}\right) - \mathbf{w_i} \tag{2.14}$$

If we want to guarantee the topological correctness of the embedding, an epsilon bound of any kind is inadequate as the only termination criterion. Instead, the process is finished only when a valid embedding is found. The embedding is valid, if and only if all faces are oriented the same, i.e. the side that was on the outside of the model is on the outside of the sphere (obviously, the surface cannot fold back upon itself without at least one triangle being upside down).

We can check this condition by testing the orientation of three consecutive vertices along the boundary of each face. Here, orientation refers to whether the three vertices make a clockwise turn on the surface of the sphere. This can be computed by evaluating $sgn\left((\mathbf{w_0} \times \mathbf{w_1}) \cdot \mathbf{w_2}\right)$. So, the relaxation is not terminated until the orientation of each face is the same as in the original model. Because this test is rather expensive, it should be done only every R iterations. We use $R = 10000$. After the embedding is valid a conventional epsilon bound is used as the final termination criterion. Actually, we use $\max_i\left(\|\mathbf{p_i}\|\right) < \epsilon$.

A relaxation process for the polyhedral model of a horse is depicted in Figure 2.4 and resulting embeddings for several models are shown in Figure 2.5.

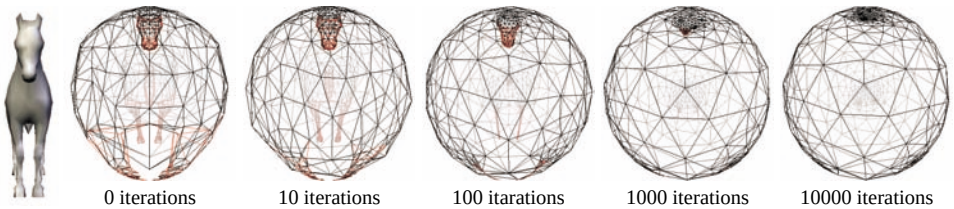


Figure 2.4: Embedding a polyhedral object on a sphere using relaxation. Initially, the vertices are projected through an interior point of the model onto a unit sphere. The relaxation is finished when all faces are oriented correctly. Incorrectly oriented faces are surrounded by red edges.

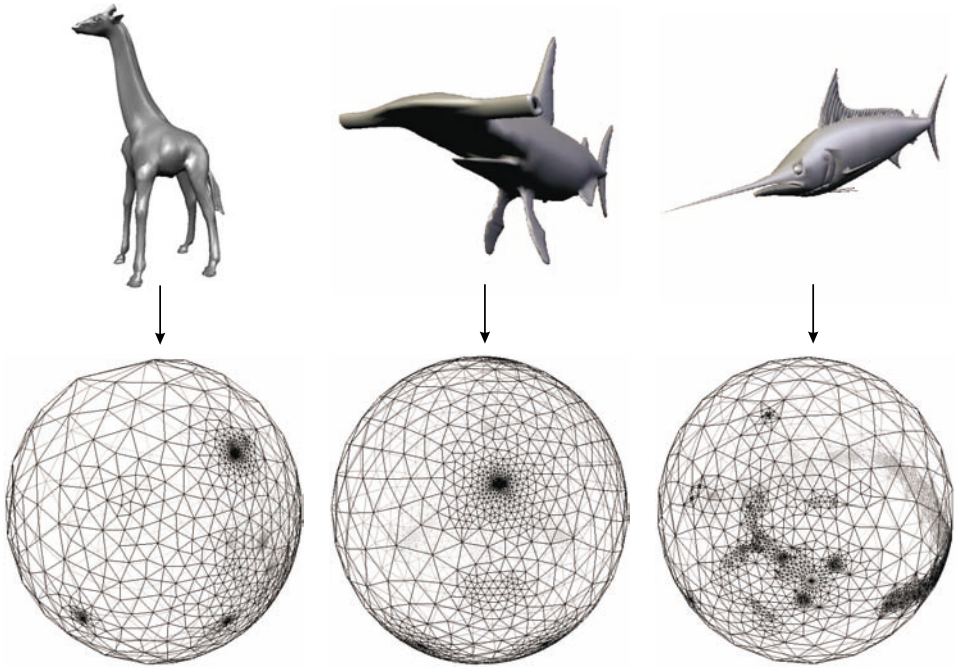


Figure 2.5: Sphere embeddings of the models of a giraffe, a hammerhead shark, and a swordfish.

2.2.6 Embedding in the plane

It is long known that the topology of the sphere can be embedded into the 2d plane. Kanai et al. [1997, 1998] use harmonic mappings to embed genus zero meshes in the plane. To find the parameterization in the plane they use the discrete harmonic mappings (see Section 2.1).

Comparison and Conclusion

None of the techniques discussed above makes a particular attempt to preserve the properties of the original mesh. Additional constraints (as discussed in Section 2.4) are necessary to make these embeddings useful. The central projection is obviously limited to a small class of objects. We find that the two techniques for general genus 0 meshes have somewhat complementary features/problems: The simplification approach is more robust (in terms of geometric computations and sensitivity to topological defects in the mesh) while the relaxation generates smooth embeddings. Both techniques suffer from the area compression problem mentioned earlier.

2.3 Isomorphic dissection

The more general approach to establish correspondence between meshes is to dissect them into pieces. Each piece is a topological disk and can be mapped to a to the plane using one of the techniques discussed in Section 2.1. Of course, the shapes have to be split in such a way that the graphs representing the dissections have equivalent topologies.

This approach is not limited to a particular topology of the shapes, since the dissection results in a set of topological disks. However, the shapes need to be homeomorphic so that their dissections could be topologically equivalent. With extra conditions it is possible to deal also with topologically different shapes.

2.3.1 Automatic dissection of shapes

Ideally, the dissection process would not require the user to assist. However, the fully automatic dissection of two meshes into isomorphic structures seems to be a hard problem. The approach of Kanai et al. [1997, 1998] uses a single patch and, thus, automatically decomposes into isomorphic structures. However, the approach is limited to genus 0 meshes and suffers from the already mentioned area compression problems in the embedding.

Several techniques exist for the dissection of a single mesh. In the context of multi resolution models several approaches require the mesh to be broken into patches. This problem is known as mesh partitioning and naturally related to graph theory. Some algorithms try to balance the size of patches (e.g., Eck et al. [1995], Karypis & Kumar [1998]).

In many multi resolution methods, however, the base domain (the structure of large patches) is found by simplifying the mesh using vertex removal [Schroeder et al. 1992; Schroeder 1997; Klein 1998; Kobbelt et al. 1998] or edge collapse [Hoppe 1996; Garland & Heckbert 1997; Lindstrom & Turk 1998].

These techniques might help in deriving a single base domain for two meshes. Lee et al. [1999] use two independently established base domain to generate one base domain for both meshes. They employ their MAPS scheme [Lee et al. 1998] to build independent parameterizations over different base domains. These base domains are merged (see Section 4) so that the resulting merged base domain contains both independent base domains as subgraphs. Note, that in general the correspondence problem had to be solved for the geometry of the base domains. Lee et al. assume that the geometry of the base domains is so similar that this problem could be solved with simple heuristics (e.g. projecting in normal direction).

2.3.2 User specification of isomorphic dissections

The underlying idea of all works in this section is that the user specifies the topology/connectivity of the base domain and the location of the base domain vertices

on the original meshes. Tracing the edges of the base domain on the mesh is more or less done automatically.

DeCarlo and Gallier [1996] do not assist the user specifying the edges. While this way of defining the dissection gives a lot of freedom to the user it is very time consuming.

Gregory et al. [1998, 1999] assist the user in defining the edges (see Figure 2.6). The base domain is developed while intersecting the surfaces. The user defines a pair of vertices on a mesh and the system finds a shortest path of mesh vertices connecting the defined vertices. Subsequently, feature vertices can be connected to existing feature vertices using shortest paths along the mesh. By picking corresponding vertices in the input meshes the system will construct the same graph in the input meshes. A problem could arise from the fact that only mesh vertices are used to find shortest path.

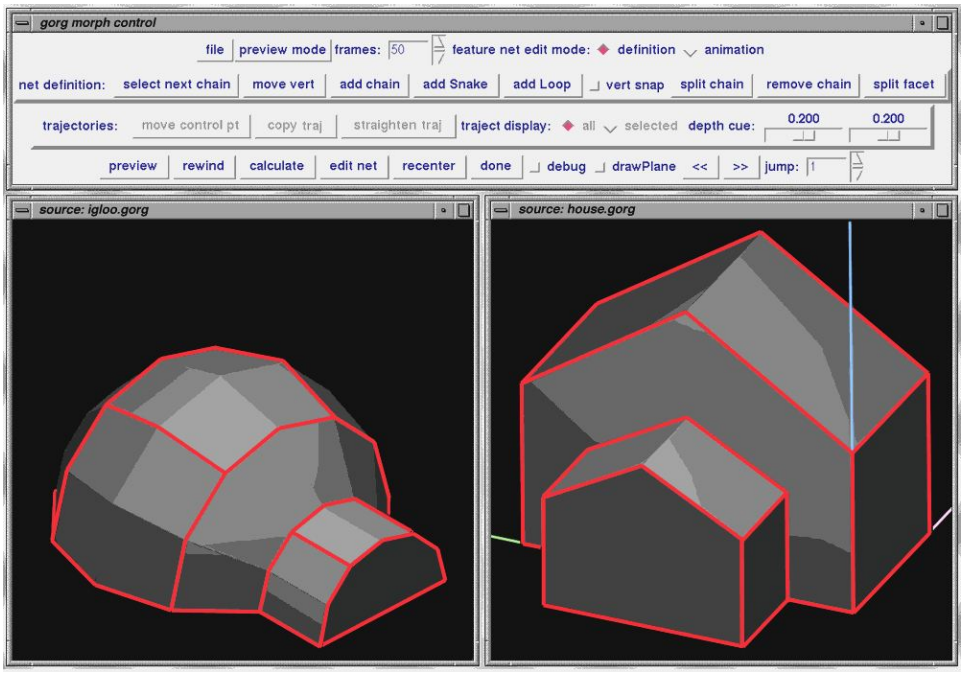


Figure 2.6: User guided decomposition of meshes. Here, the user constructs closed loops and segments to dissect two meshes into isomorphic patch networks. Reprinted from Gregory et al. [1998].

The works of Bao & Peng [1998] and Zöckler et al. [2000] are similar in spirit. However, it seems that they allow to use more points to define the boundary of a patch. Points are connected with the shortest paths in the vertex-edge graph as in the work of Gregory et al. [1998, 1999]

In the approach of Kanai et al. [2000] the user first defines a set of corresponding feature vertices. Aware of the problems resulting from using a shortest path

consisting of mesh vertices the authors compute the shortest path on the piecewise linear surface connecting the feature vertices. This path may or may not coincide with vertices and edges. Figure 2.7 shows the resulting dissection for two cars. Since computing exact shortest path on polyhedral surfaces is difficult and time consuming they employ an approximate method that refines the original mesh and uses Dijkstra's algorithm [Kanai & Suzuki 2000].

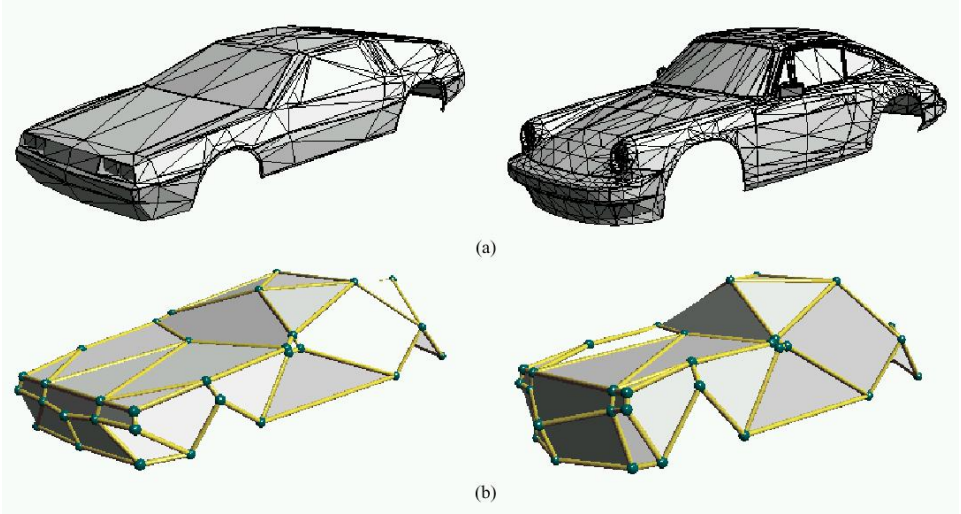


Figure 2.7: In the approach of Kanai et al. [2000] the user selects only corresponding vertices and how they are to be connected. The mesh is the dissected using shortest path connecting the vertices. Reprinted from Kanai et al. [2000].

However, even using the exact shortest path can lead to problems. Praun et al. [2001] illustrate the problem and propose better solutions: If a shortest path would cross an already established edge of the base domain, the shortest possible connection avoiding the intersection is computed using a wavefront algorithm. However, also the order of vertices being connected is important, because several edges might enclose an unconnected vertex. This problem can be avoided by traversing the vertices along a spanning tree.

In our view, the underlying problem is that on non-convex and unbounded shapes more than one geodesic between two points exists on the surface. We believe that a set of these geodesics is sufficient to trace out the given connectivity of the base domain. To implement this, first all geodesics between connected vertices of the base domain would be computed. Then, these edges would be inspected for possible intersection. The intersection-free subset yields the decomposition of the original mesh.

2.4 Feature alignment

The necessity for aligning prominent features becomes evident even in very simple examples. Figure 2.8 shows two morphs between models of a young pig and a grown-up pig. In the upper sequence, no features were aligned and the resulting morph is unacceptable. The lower sequence of Figure 2.8 shows a morph produced with some features (ears, eyes, hoofs, and the tail) aligned. The result is obviously more pleasing. Surprisingly, the need of user guidance becomes more obvious when the shapes are similar. This is because we can envision a transformation, i.e. we expect common features of the models (head, legs, etc.) to be preserved. But this does not happen, of course, due to the different mesh topology of the models (in this example, the different mesh topologies are obvious from the different vertex counts of the models).

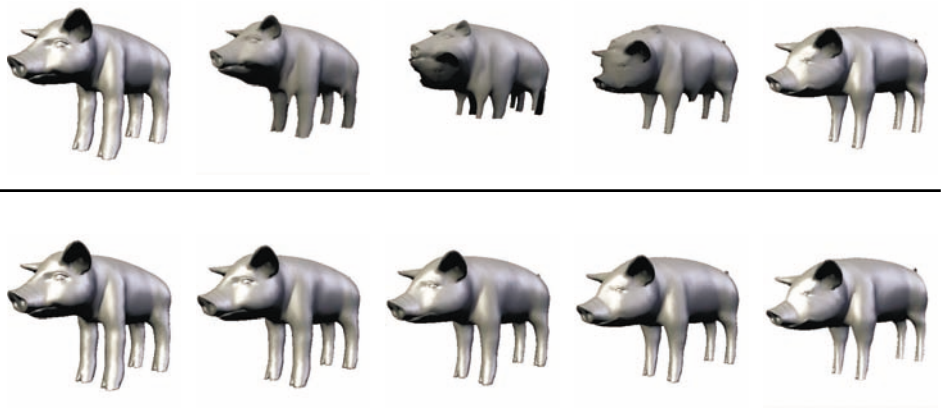


Figure 2.8: Morphs between the models of a young pig and a grown-up pig. In the upper row, no feature alignment is used, which leads to unpleasant effects (e.g., eight legs in the intermediate models). In the lower row, the eyes, ears, hoofs, and the tail are aligned (a total of 17 vertex-vertex correspondences), yielding a smooth transformation.

2.4.1 User-selected vs. shape features

A difficult task is to identify common features of several shapes. It seems impossible to automatically find such common features as they are mostly defined in a semantic and not necessarily in a geometric way. The user can identify these features and provide information about their location and correspondence (for instance as vertex-vertex correspondence of a few vertices). The algorithm should exploit this information as much as possible.

All dissection type methods explained above offer this way of user-control. Since the user explicitly chooses corresponding patches (and, therefore, corre-

sponding edges and vertices) they can specify which parts of the meshes correspond. However, except for the techniques presented by Kanai et al. the user is also involved in other tasks, which make the process complicated and lengthy.

However, the shapes' geometries also contain information useful to exploit. One could extract prominent geometric features of the shapes (e.g. Hubeli & Gross [2001]) and try to match them. Several functions over the parameter domain of the function seem to be worth looking at. It is important that these functions are independent of the parameterization, i.e. are intrinsic to the shape and do not change if the description of the shape is changed. Such functions are especially considered in *differential geometry*, which could be seen as exploring a shape on the shape, without a distant view. The most prominent assets for describing shapes in differential geometry are

- normals (which are independent of translation and scaling but sensitive to rotation) and
- curvature (principal curvatures,mean or Gaussian curvature), which is independent of translation and rotation but sensitive to scaling.

The parameterization of the shape's boundary allows to represent these quantities as a function in two variables, i.e. the normal $n : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ or the Gaussian curvature $c : \mathbb{R}^2 \rightarrow \mathbb{R}$. More generally, these descriptions are part of the fundamental forms of the approximated smooth surface.

It is clear that this information about the shapes does not lead to point to point correspondences such as user selected features. Instead the quality of the match of two shapes is quantified as a function of the distance of the shape descriptors. For example, Surahzky and Elber [2001] use the Integral over the inner products of normals:

$$D = \int_S \langle n_1, n_2 \rangle \, dS \tag{2.15}$$

Here, the inner product between normals and the integration over the surface represent particular choices. One might choose another metric for the difference of normals as well as another method to take into account the set of differences (e.g. the maximum of the angles between normals). In order to match shapes based on such criteria the parameterization is changed so that the functional is minimized. Note that no point has an a-priori optimum placement making this problem much harder to solve than aligning specified point to point correspondences.

2.4.2 Transforming to align features

As a first step in an alignment procedure the parameter domains should be transformed using affine transform to roughly align the features. Note that this is not possible for parameterizations resulting from dissection as the orientation of each patch is determined by neighboring patches.

In [Alexa 1999; Alexa 2000] we align a set of point to point correspondences by rotating the spherical embeddings of the mesh. The objective function to be minimized is the squared distance of corresponding points, however, we could have also used the inner products of points on the sphere. The minimization problem can be solved using multidimensional numerical minimization (three values have to be found: an axis of rotation and an angle).

2.4.3 Warping parameterizations to align features

In general, one could generate any parameterization of the meshes as a first step to establish correspondence. After this, the parameterization domain can be used to align user selected features or automatically generated features in terms of a re-parameterization of one or more of the initial parameterizations.

Zöckler et al. [2000] and our approaches [Alexa 1999; Alexa 2000] allow the user to select a set of point to point correspondences. Warping techniques similar to those used in image morphing (e.g., see [Ruprecht & Muller 1995; Wolberg 1998]) are used deform the parameterization so that cooresponding points coincide. Whether the parameter domain is a disk as in [Zöckler et al. 2000] or a sphere as in [Alexa 1999; Alexa 2000] does not make a difference for the general approach.

In contrast to image warping, it is absolutely necessary that the warp does not introduce incorrectly oriented faces, This would be less of a problem, if vertices as well as edges would be warped. But since the algorithm later requires edge-edge intersection tests, warping the edges is impractical. Instead, edges should be (still) defined as the shortest path between vertices. That is, we warp the vertices only. Thus, even injective warping functions might introduce fold-over.

The solution is to make several local and small deformations instead of searching for one warping function that is applied once to all vertices. This way, we can control the effect of the deformation (does it introduce fold-over, or not). The mapping function proposed in [Alexa 1999; Alexa 2000] to move a vertex from $\mathbf{w}$ to $\hat{\mathbf{w}}$ is

$$f(\mathbf{x}) = \begin{cases} \mathbf{x} + c\,(r - \|\mathbf{x} - \mathbf{w}\|)\,(\hat{\mathbf{w}} - \mathbf{w}) & \|\mathbf{x} - \mathbf{w}\| < r \\ \mathbf{x} & \|\mathbf{x} - \mathbf{w}\| \geq r \end{cases} \tag{2.16}$$

where r is the radius of influence for the map. However, other functions could be used as well.

We have proposed to warp only as much as is possible with the given triangulation. If the mapping starts to introduce fold-over in the triangulation we first shorten the move length by adjusting the constant c and then make the map more local by adjusting the radius of influence r. However, the features are not guaranteed to coincide after this process.

The map is applied to both models and the necessary move is derived from the current positions. That is, we do not define an intermediate position for the feature vertices, as is common in image morphing. So, in case the deformation in one

model would introduce fold-overs, coincidence can still be achieved by deforming the other model.

Zöckler et al use the fold-over free warping scheme of Fujimura and Makarov [1998]. They also warp in small steps. However, if fold-over occurs they change the mesh topology to assure that the embedding stays valid. In particular, they use edge flips for this task. This changes the original triangulation of the meshes.

Recent work in texture mapping allows to incorporate point constraints [Eckstein et al. 2001; Lévy 2001]. These techniques could be applied for the problem here.

Lévy [2001] formulates the problem of satisfying given point constraints by incorporating the squared error of the point correspondences into the energy functional used to generate the parameterization. Using a scalar to weigh the importance of the point correspondence allows to trade between the regularization term for the smoothness of the parameterization and the accuracy of satisfying the constraints. On the other hand this mixed energy functional does not guarantee a valid embedding. A possible way would be to start with a valid embedding and then increasing importance of the constraints as long as the embedding stays valid.

Eckstein et al. [2001] propose a scheme that allows to exactly satisfy constraints whenever possible. It might be necessary to introduce additional vertices in the triangulation for this. The triangulation is first simplified so that it contains only the constrained vertices. These are placed accordingly and the mesh is then refined again. During the refinement process it might be necessary to insert additional vertices because straight edges connecting vertices could intersect.

2.5 Conclusions

The ideal algorithm for finding a parameterization of a mesh has not been found. In general, coarse simplifications of the original meshes are accepted as useful parameter domains. In the context of morphing they are not ideal for two reasons:

- For seemingly different shapes a common base domain might be hard to find and the decomposition of the original mesh *forces* the user to interact.

- The alignment of features (e.g. shape features) is restricted to corresponding patches of the base domain.

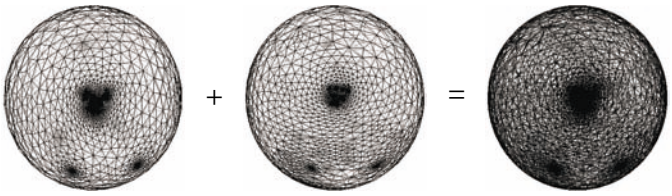
In view of these limitations the simple solution to embed topological spheres on a unit sphere has some appeal. However, embedding complex shapes on a sphere might result in a distorted parameterization because the local ratio of surface area between sphere and original shape differs.

It seems that finding a common base domain is the method of choice. For applications, in which one base domain is needed for more than one shape, techniques should be developed that include geometric features in the decomposition process.

We still search for a reliable method that works on arbitrary input, takes any number of user-constraints into account, optimizes a reasonable resemblance of the shapes, and is sufficiently fast.

Chapter 3

Constructing representations



Given two embeddings W_0, W_1 of meshes $(V_0, K_0), (V_1, K_1)$ on a common domain D we aim at generating one mesh connectivity K with vertex positions $V(0)$,$V(1)$ so that the original shapes are reproduced, i.e.

$$\phi_V(0)(|K|) = \phi_{V_0}(|K_0|), \phi_V(1)(|K|) = \phi_{V_1}(|K_1|). \tag{3.1}$$

Note that the vertex positions $V(0), V(1)$ are already available using the barycentric coordinates of each vertex w.r.t. the base domain. These barycentric coordinates allow to map each vertex from one mesh to the other. However, the exact mapping of vertices onto the piecewise linear surface might lead to bad results. The next subsection discusses better alternatives for the absolute position of vertices.

The main point of this chapter is to establish the common connectivity K. The typical approach found in the morphing literature is to generate a supergraph of the connectivities K_0, K_1, i.e. one that contains the simplices of both plus additional vertices if edges cross. This graph is found by *map overlay*. Here, we distinguish two cases:

1. Bounded meshes embedded in a disk.

2. Unbounded meshes, assuming the geometries of several meshes are sufficiently close.

These cases stem from the parameterization methods presented in the previous chapter.

Looking at multiresolution techniques for meshes is an alternative way of generating a common connectivity is remeshing. In particular, the parameterization is exploited to map planar coordinates of refinement operators to coordinates on the surface of the shapes. Provided the base domain accurately represents sharp features of the meshes this approach has the advantage that it is much easier to scale. The size can be easily adapted to the desired precision. For the same reason this approach is easier to extend to more than two meshes.

3.1 Mapping parameter values to the surface

After the meshes have been parameterized it is easy to find the position of a particular vertex on the surface of a mesh. Assume we want to find the position of vertex $\mathbf{v_{1_i}}$ of the first mesh on the second mesh. We determine the vertices $\{\mathbf{w_{2_j}}\}$ comprising the face in the parameterization in which the parameter domain position $\mathbf{w_{1_i}}$ lies. Then, $\mathbf{w_{1_i}}$ is represented in barycentric coordinates with respect to $\{\mathbf{w_{2_j}}\}$:

$$\mathbf{w_{1_i}} = \sum b_k \mathbf{w_{2_{j(k)}}} \tag{3.2}$$

The position of $\mathbf{v_{1_i}}$ in the other mesh is found as

$$\mathbf{w_{1_i}}' = \sum b_k \mathbf{v_{2_{j(k)}}} \tag{3.3}$$

This is the exact position on the piecewise linear shape and the way used in most of the morphing literature.

However, this does not take into account the idea that piecewise linear shapes are (in most cases) just approximations of smooth shapes. Particular practical problems occur when normals have to be rebuild from these new geometric positions: Vertices inside a face get the face's normal. If standard rendering methods are used (vertex normals and Gouraud shading) this results in degenerate shading.

It would be advantageous to find positions which result in a smooth surface. More specifically, we would like to use the barycentric coordinates to find positions *over* a triangular face and not necessarily on the face. This calls for methods defining a smooth surface from a coarse mesh. An obvious choice for such a method would be subdivision (e.g., Loop subdivision [Loop & DeRose 1990] or Kobbelts $\sqrt{3}$-scheme [Kobbelt 2000]).

3.2 Map overlay data structure

We need a data structure to store the meshes, which allows to add and remove edges, gives quick access to topological information (e.g., the ordering of edges around a vertex), and is not too heavy in terms of storage. We choose the doubly connected edge list [Muller & Preparata 1978] (sometimes called twin-edge data structure). The basic data type of this data structure is the edge. Edges are stored as two directed half edges. More specifically, the following information is stored:

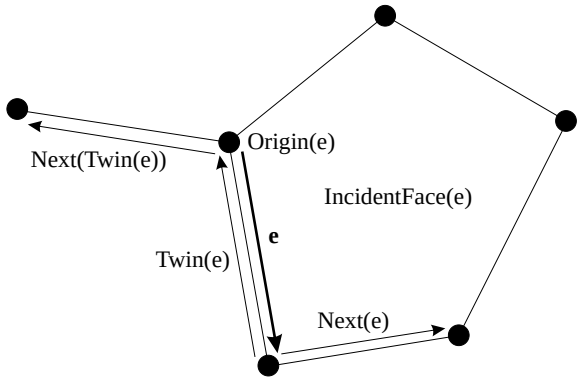


Figure 3.1: The doubly connected edge list.

Face The face record contains a pointer to an arbitrary half edge on its boundary.

Edge Each edge record contains pointers to

- its originating vertex,
- the face it bounds,
- the half edge connecting the same vertices but in the opposite direction (its *twin*),
- the next half edge along the boundary of the bounded face.

Vertex The vertex record contains a pointer to an arbitrary half edge originating from this vertex as well as location in space and other attributes (e.g., normal, color, texture coordinate).

Figure3.1 illustrates the data structure. Note that it is particular easy to iterate along the boundaries of faces (next pointers) or through all edges incident upon a vertex in their circular order (twin $\rightarrow$ next). A good description of the doubly connected edge list can be found in de Berg et al. [1997].

3.3 Open meshes embedded in a disk

Several algorithms were proposed for the problem of overlaying planar graphs - see a textbook [de Berg et al. 1997]. In general, the planar map overlay has the complexity $O(n \log n + k)$, where n is the number of edges and k is the number of intersections. If the two subdivisions are connected (as in our case) the planar overlay can be computed in $O(n + k)$ [Finke & Hinrichs 1995].

The general paradigm for planar overlay is *plane sweep*. Sweep algorithms process the input with a virtual line moving along its normal direction. Whenever a vertex intersects the sweep line the corresponding edge is added (the vertex is the

starting point of this edge) or removed (the vertex is the endpoint) from the list of active edges. The list of active edges is tested for intersection with added edges. To further reduce the number of necessary intersection tests the active edges are stored in their order along the sweep line. This is done by inserting edges in the correct position. In addition, the order has to be updated at intersection points. Using the ordering, only neighboring edges have to be tested for intersection. This processing leads to an algorithm with complexity $O(n \log n + k)$. By exploiting that two connected graphs are intersected the complexity can be reduced to $O(n + k)$.

In the case that meshes are embedded on the disk special care has to be taken for the boundaries of the meshes. While we assume that the embedding is surjective (i.e. fills the disk), the boundary in fact is a polygon leaving small empty regions between the disk and the polygon. However, it is clear that the boundaries of the meshes to overlay should be mapped onto each other. So in order to avoid that the boundary polygons intersect with inner edges of the other mesh the boundaries have to be merged first. This is done by simply connecting the vertices of all meshes on the disk along the linear order given by the disks boundary. After this boundary polygon has been established the planar mesh overlay procedure can be computed.

3.4 Closed meshes in arbitrary position

There seem to be only a few publications about the overlay of meshes in general position (i.e. the triangulated surfaces are close to each other but not e.g. planar). Note that plane sweep solutions are not applicable in this case. Few publications deal with overlaying two subdivision of the sphere. Kent et al. [1992] give an algorithm for the sphere overlay problem, which needs $O(n + k \log k)$ time. We have presented a solution to this particular problem, which *reports* the intersection of two spherical subdivisions in the optimum time of $O(n + k)$ [Alexa 2000]. Also, both algorithms exploit the topological properties of both subdivisions, which are used to guarantee the correct order of intersections. Here, we generalize these algorithms to work on two arbitrary shaped meshes, which are assumed to be sufficiently close to each other. We also alleviate the problem that the published version [Alexa 2000] had a worst case complexity of $O(n + k \log n)$ for the *construction* of the merged mesh using the already reported intersections.

The algorithm consists of two main parts: First, finding all intersections, and second, constructing a representation for the merged model.

3.5 Finding the intersections

In the algorithm two geometric functions are needed: One to decide if and where two edges intersect on the sphere, and a second to decide whether a point lies inside a face. Both geometric properties can be checked in a projection to the tangent

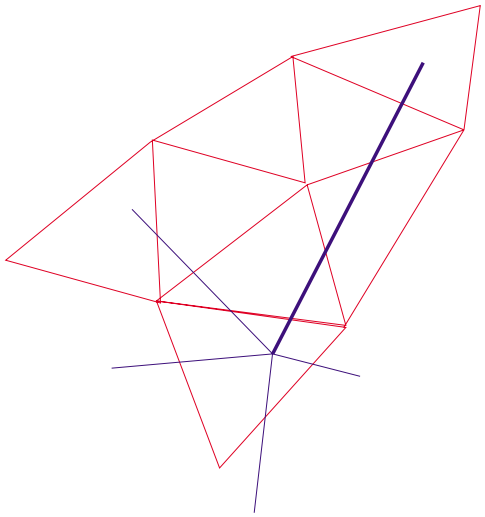


Figure 3.2: Edge-edge intersections are determined by following an edge (blue in this illustration) over the faces of the other triangulation (red). After finding an intersection the face-to-face coherence exploited and only the edges of the next face are tested.

plane of the surfaces. Since the meshes are supposed to be close in space their tangent planes should not differ to much. A suitable way of finding a common tangent plane is to take the cross product of two edges (i.e. the two edges to intersect, or two edges of the face to check).

The basic idea is to traverse the graphs breadth first, keeping information about the face that contains the current working edge and exploiting face-to-face neighbor information. Choose an arbitrary vertex $\{i\} \in K_0$ and search the 2-simplex $f = \{f_1, f_2, f_3\} \in K_1$ that contains it in under the bijective mapping. Start with an edge $e \in \mathcal{S}(i)$. Store e together with f on a stack. In general, the stack will always contain a directed edge together with the face in the other mesh containing the origin of this edge. The basic idea of the traversal is to walk over the faces following an edge (see Figure 3.2). Each edge $e = \{e_1, e_2\}$ is intersected first with the three edges $\{f_1, f_2\}, \{f_2, f_3\}, \{f_3, f_1\}$ bounding f, which contains $\phi_{W_0}(e_1)$. When an intersection is found the working edge e is emanating to the next face, i.e. the one that shares the intersected edge. This face is set to be f and is inspected in turn.

The same process is repeated with edges in K_1. This is necessary to find the topological orders of edges in K_0 cutting edges in K_1. Each edge is tested against three edges plus two additional intersection tests for each intersection being found. Thus, the algorithm has constant costs per edge and per intersection and the complexity is $O(n + k)$.

3.6 Generating the data structures

An appropriate data structure for storing the intersections is needed. Information about an intersection should be accessible from both intersecting edges at constant costs. We use a hashtable with edge indices as key values. When edge-edge intersections are found and stored in the intersection lists a pointer to the entry in the hashtable is stored. This means, both edges point to the same data structure containing information about the intersection (the intersecting edges in the beginning). The hashtable is only needed to access the entry when the intersections have already been computed by processing K_0 and need to be found when intersections from K_1 are generated. After reporting all intersections the hashtable is discarded.

The following two step algorithm constructs the merged mesh: First, edges in K_0 are cut. We iterate through the intersection list of an edge and cut the edge at each intersection point. Thus, a new edge (two half edges) are generated for each intersection. The new edge represents the part of the edge that has to be processed. At each intersection the data structure containing the respective information is updated to now contain the two parts of the edge incident upon the intersection point. At this point only the twin pointers of the half edges are updated. The next pointers are left empty.

Second, edges in K_1 are processed. As in the first step edges are cut into two pieces at each intersection point. However, this time also the next pointers are updated. This is done by using the information stored in the intersection data structure, which now contains both edges of the already cut edge in K_1.

After all intersections are processed in this way we have a valid vertex and edge lists of the embedding. It remains to compute the records for the faces. Note that faces created from intersecting triangles are convex polygons with 3 to 6 sides, which should be triangulated. This is another subtlety, which is more involved as it may seem: While the polygon resulting from the intersection is convex it is not clear what shape it has in other geometric configurations, e.g. those of the source meshes. In principle one should find a triangulation that is admissible in all source geometries. This might be difficult and could lead to the need for additional vertices. The problem is known as *compatible triangulation* and discussed in detail in another context in Section 4.4.1.

Note that it has been communicated that this approach could be extended to bounded meshes, however, boundaries require special treatment beyond the scope of this work.

3.7 Remeshing

A mesh is typically just an approximation of a shape. We have already seen that the mesh overlay process together with using coordinates lying exactly on the mesh might introduce artifacts into the source meshes (see Section 4.1). Thus, even if the original mesh connectivities are available as subsets of K the reproduction of the

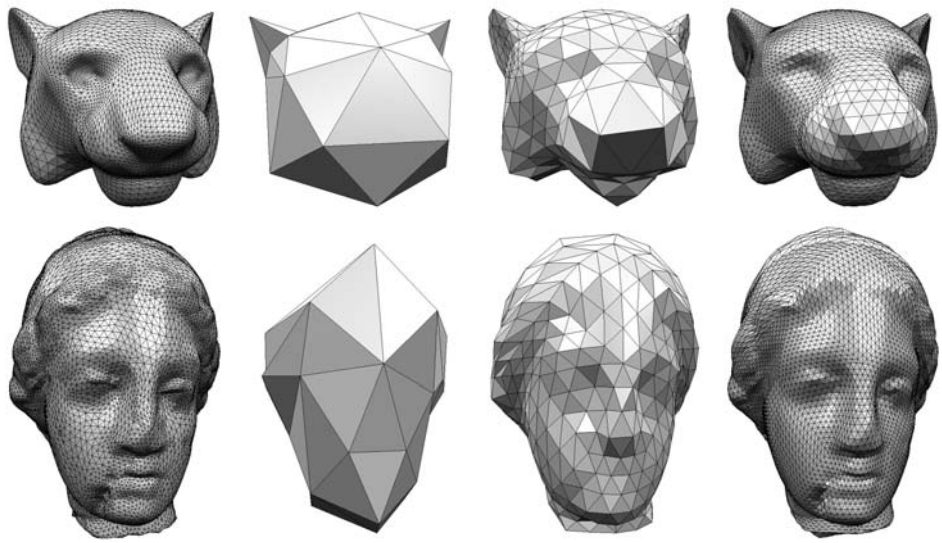


Figure 3.3: A multiresolution mesh representation build over the same base domain to represent two geometries. Reprinted from Michikawa et al. [2001].

original shapes though exact is not ideal. It seems that the perfect reconstruction of the source shapes is impossible and we could as well use any mesh connectivity to approximate both given shapes.

Remeshing techniques have been used to construct semi-regular meshes from irregular input [Lee et al. 1998]. The irregular mesh is reduced to an irregular base domain. The base domain is refined inserting only regular vertices. The idea is to use refinement operators as known from subdivision surfaces, however, without using the geometric rules attached to the refinement. Instead, geometric positions are found by exploiting the bijection between original surface geometry and the parameterization. For example, using the 1-4 split the parameter domain positions of inserted vertices are given as edge bisectors. This parameter leads to the coordinate on the surface of the mesh.

This idea has recently been used to construct morphable meshes by Michikawa et al. [2001] (see Figure 3.3). In this context, each parameter value leads to two coordinates. After several refinement steps a semi-regular mesh connectivity K is constructed together with coordinates $V(0), V(1)$ as desired. Because the refined connectivity is defined by the rules of the refinement used, only the base domain connectivity has to be stored explicitly.

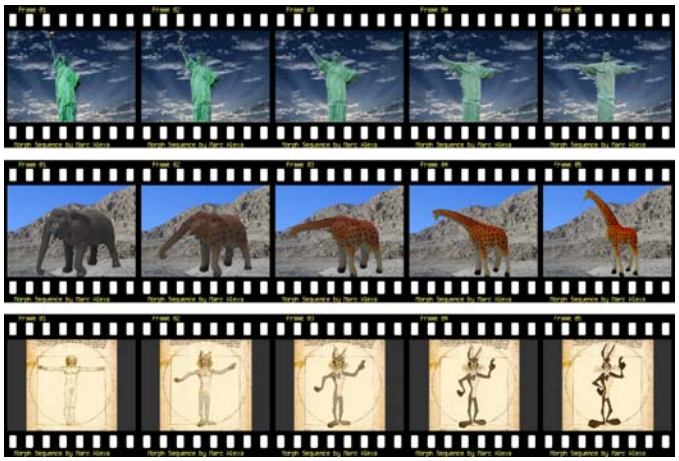
To achieve a desired approximation accuracy, the number of refinement steps should be adapted to the geometric complexity of the meshes. Note that refinement could be done adaptively depending on the viewing conditions without necessarily computing and storing all coordinates of the refinement levels.

3.8 Comments

The remeshing approach is appealing because it allows to scale the size of the representation mesh. Its only limitation is the accurate representation of sharp features in the original shapes. In conventional multiresolution models this problem is alleviated by fitting the base domain to these features. In the context of morphing the base domain has to represent the features of several meshes, which do not necessarily coincide. This, again, incurs extra burden on the user, because a more complex base domain has to be induced on the input meshes. In addition, a more complex base domain limits the possibilities of automatic feature alignment methods. However, the flexible and lean representation mesh seems worth it.

Chapter 4

Interpolating corresponding shapes



Using the methods presented in the previous chapters, it is possible to generate one topological shape representation for several shapes. Topology, in this context, refers to the number and structure of primitives used to describe the shape, e.g. the number of line segments in a polygon or the connectivity information of a mesh. In any case, the shape is represented as the coordinates of vertices.

A morph sequence contains several states of an object transforming from one state to another. The natural idea to generate morphs is to interpolate representations of shapes. The easiest way is to interpolate coordinates of vertices as they are readily available. However, as with corresponding features of the shapes, the human observer has certain expectations regarding interpolated shapes. It is difficult to define a set of rules which have to be followed. Different approaches to the interpolation of shapes are characterized by different conditions, which are believed to describe natural shape interpolation.

The coefficients describing such linear interpolation are sometimes called transition parameters. In classical morphing applications, the transition parameter can be represented by a single scalar ranging from 0 (representing the source shape) to 1 (representing the target shape) and is mentally connected to time values in an animated sequence.

4.1 Linear Interpolation of Boundary and/or Orientation

The easiest way to produce blends of corresponding shapes is to interpolate the coordinates of vertices. Assume a set of n shapes is represented by a topological structure $\mathcal{S}$ and vectors V_i containing vertex coordinates. $\mathcal{S}$ might be polygon, a skeleton, a triangulation, a mesh, a tetrahedralization, or some other structural description of – at least – the shape's boundary. The V_i's just contain real numbers.

Given a transition parameter t_i the coordinates of an interpolated shape are computed by

$$V = \sum_i t_i V_i \tag{4.1}$$

This is the easiest way of computing interpolated shapes. It produces good results if the shapes have the same orientation and are somewhat similar. Figures 4.1, 4.2, and 4.3 show morph sequences obtained by linear interpolation.

Different orientation could lead to displeasing results. Imagine two squares that are rotated by 180 degrees against each other. If simple vertex interpolation is applied in this configuration, the interpolated shapes will shrink until the shape is collapsed to one point and then grow again. This is not the desired result in most applications. It is advisable to interpolate the orientation separately from the vertex coordinates.

4.1.1 Interpolation of orientation

Several ways exist to compute a relative orientation of two shapes. Note that it is difficult to interpolate the orientation of more than two shapes in 3D so the following discussion will be restricted to two shapes.

As a first step, the shapes are usually translated so that their centers of mass coincide with the origin. Then, a rotation [Cohen-Or & Carmel 1998; Cohen-Or et al. 1998] or an affine transform [Alexa 2000] is computed seprating the rigid/affine part from the elastic part of the morph. A way of defining the rigid/affine part is to minimize the squared distances of corresponding vertices using the corresponding transform. The minimization problem of finding an affine transform can be solved using the pseudo inverse of the coordinate vector. Let the vertex vectors be

arranged as a $n \times 3$ matrix

$$V = \begin{pmatrix} v_{1_x} & v_{1_y} & v_{1_z} \\ v_{2_x} & v_{2_y} & v_{2_z} \\ v_{3_x} & v_{3_y} & v_{3_z} \\ \dots & & \end{pmatrix}.$$

Then the squared distance of coordinates under an affine transform A is

$$(V(0)A - V(1))^2 \tag{4.2}$$

and has to be minimized. This leads to linear system of equations, which can be solved using pseudo inverse $V(0)^+$:

$$A = V(0)^+ V(1) = \left(V(0)^T V(0)\right)^{-1} V(0)^T V(1) \tag{4.3}$$

Alternatively, the least squares solution (or, the pseudo inverse) could be computed using the SVD, which allows explicit control over the sensitivity to near rank deficiencies [Golub & Van Loan 1989].

Intermediate shapes $V(t) = \{\mathbf{v_1}(t), \mathbf{v_2}(t), \dots\}$ are described as $V(t) = A(t)V(0)$. The question is how to define $A(t)$ reasonably? The simplest solution would be: $A(t) = (1-t)I + tA$. However, some properties of $A(t)$ seem to be desirable, calling for a more elaborate approach:

- The transformation should be symmetric.
- The rotational angle(s) and scale should change monotonic.
- The transform should not reflect.
- The resulting paths should be simple.

The basic idea is to factor A into rotations (orthogonal matrices) and scale-shear parts with positive scaling components. We have examined several decompositions [Alexa et al. 2000]. Through experimentation, wehave found a decomposition into a single rotation and a symmetric matrix (i.e. the polar decomposition), to yield the visually-best transformations. This result is supported by Shoemake & Duff [1992] for mathematical, as well as psychological, reasons. The decomposition can be deduced from the SVD as follows

$$\begin{aligned} A &= R_\alpha D R_\beta = R_\alpha (R_\beta R_\beta^T) D R_\beta = \\ &\quad (R_\alpha R_\beta)(R_\beta^T D R_\beta) = R_\gamma S \end{aligned} \tag{4.4}$$

however, there are computationally cheaper alternatives [Shoemake & Duff 1992]. Based on the decomposition, $A(t)$ is computed by linearly interpolating the free parameters in the factorizations in (4.4) , i.e.

$$A_\gamma(t) = R_{t\gamma}((1-t)I + tS). \tag{4.5}$$

Figure 4.4 illustrates the resulting transformations for a triangle. For comparison, 4.4(a) shows linear interpolation of vertex coordinates. The transformation resulting from a singular value decomposition and linear interpolation $A_{\alpha,\beta}(t)$ is depicted in 4.4(b). Note that the result is symmetric and linear in the rotation angle but still unsatisfactory, since a rotation of more than π is unnecessary. However, if we subtract 2π from one of the angles (depicted in 4.4(c)) the result is even more displeasing. We have found that decomposing A into one rotation and a symmetric matrix and using $A_\gamma(t)$ yields the best results (Figure 4.4(d)). It avoids unnecessary rotation or shear compared to the SVD and is usually more symmetric than a QR decomposition-based approach.

4.2 Interpolation of intrinsic boundary representation

Linear interpolation of vertices can lead to undesirable effects such as shortening of parts of the boundary during the transition. To avoid such problems, Sederberg et al. [1993] propose to interpolate an intrinsic representation of the boundary. For polygons, such an intrinsic representation is edge length an interior angles. Unfortunately, there is no simple analogue in 3D. An attempt was made to extend the ideas of to polyhedra [Sun et al. 1997] but the methods are computationally expensive and unreliable. Here, only the 2D case will be explained.

Assume the polygons P and Q are described by their vertex positions $\mathbf{p_i}, \mathbf{q_i}$. Let $\theta_{p_i}, \theta_{Q_i}$ be the interior angles around $\mathbf{p_i}, \mathbf{q_i}$ and

$$L_{P_i} = |\mathbf{p_{i+1}} - \mathbf{p_i}|, L_{Q_i} = |\mathbf{q_{i+1}} - \mathbf{q_i}| \tag{4.6}$$

the length of the i-th edge. Additionally, let $\alpha_{P_i}, \alpha_{Q_i}$ be the angles between the i-th edge and a fixed axis. An intermediate polygon is represented by

$$L_i(t) = (1-t)L_{P_i} + L_{Q_i} \tag{4.7}$$
$$\theta_i(t) = (1-t)\theta_{P_i} + \theta_{Q_i} \tag{4.8}$$
$$\alpha_i(t) = (1-t)\alpha_{P_i} + \alpha_{Q_i} \tag{4.9}$$

However, this description will lead to an open polygon in the general case.

4.2.1 Closing the polygon

The idea is to close the polygon by small changes of the defining parameters. Since the edge length has to be changed in some cases and, on the other hand, it is easier to change only one of three parameters, only the edge lengths will be changed to close the polygon.

The interpolated edge becomes

$$L_i(t) = (1-t)L_{P_i} + L_{Q_i} + S_i \tag{4.10}$$

To uniquely determine the S_i, the squared relative length change

$$g(S_0, \ldots, S_n) = \sum_i \frac{S_i^2}{|L_{P_i} - L_{Q_i}|^2 + \epsilon}, \quad \epsilon > 0 \tag{4.11}$$

will be minimized under the constraint that the polygon has to be closed. Closure of the polygon can be formulated by the following necessary conditions:

$$\phi_1(S_0, \ldots, S_n) = \sum_i ((1-t)L_{P_i} + L_{Q_i} + S_i) \cos\alpha_i = 0 \tag{4.12}$$

$$\phi_2(S_0, \ldots, S_n) = \sum_i ((1-t)L_{P_i} + L_{Q_i} + S_i) \sin\alpha_i = 0 \tag{4.13}$$

This kind of minimization problem can be solved using Lagrange multipliers.

$$\Phi(\lambda_1, \lambda_2, S_0, \ldots, S_n) = g + \lambda_1\phi_1 + \lambda_2\phi_2 \tag{4.14}$$

This leads to the surprsingly compact solution

$$S_i = -\frac{1}{2} |L_{P_i} - L_{Q_i}|^2 (\lambda_1 \cos\alpha_i + \lambda_2 \sin\alpha_i) \tag{4.15}$$

4.3 Interpolation of differential boundary representation

In classical morphing applications the transition parameter can be represented by a single scalar ranging from 0 (representing the source shape) to one (representing the target shape) and is mentally connected to time values in an animated sequence.

Now we want to locally morph certain features or regions of interest, i.e. the transition parameters are different for different vertices. We will call the set of transition parameters for vertices the *transition state*. A major problem when morphing only locally arises from the fact that corresponding features might not have the same position in space and, thus, interpolation of absolute coordinate could lead to undesirable effects. This problem is illustrated in Figure 4.5 The shapes in a) and b) are source and target geometry of one mesh. The idea is to locally change the geometry of the baby's face so that the nose takes the shape of the boy's. Locally interpolating vertex coordinate leads to the shape depicted in c), which is clearly not usable. Note that the faces are overall aligned in space and that the misalignment of the noses results from different relative positions in the faces.

We could ease the problem of misalignment by assigning an affine transform to a local morph. However, this leads to problems when features overlap. More generally, a shape should be defined by the transition state of its vertices. In that way, the transition states is representative for the shape of a morphable object. This could be a very compact way of representing deforming or animated objects.

The main idea to overcome the mentioned problems is to represent vertex coordinates with respect to their neighbors in the mesh. Given a vertex and its one-neighborhood ring (see Figure 2 a), the position should be described relative to

the positions of vertices in the neighborhood ring. Further, the representation of a vertex should be linear in the absolute coordinates. Non-linear functions tend to be numerically difficult to handle and many morphable meshes have sliver triangles, which, together, leads to unpredictable results.

The relative representation aims at making the shape of the mesh invariant to translation or, ideally, invariant under affine transforms. If a vertex were represented in the affine space of its neighbors invariance under affine transforms would trivially follow. Floater and Gotsman have shown how to use such representations to morph planar triangulations [Floater & Gotsman 1999]. The extension to triangulations in $\mathbb{R}^3$is difficult because vertices of a the neighborhood are not necessarily affinely independent.

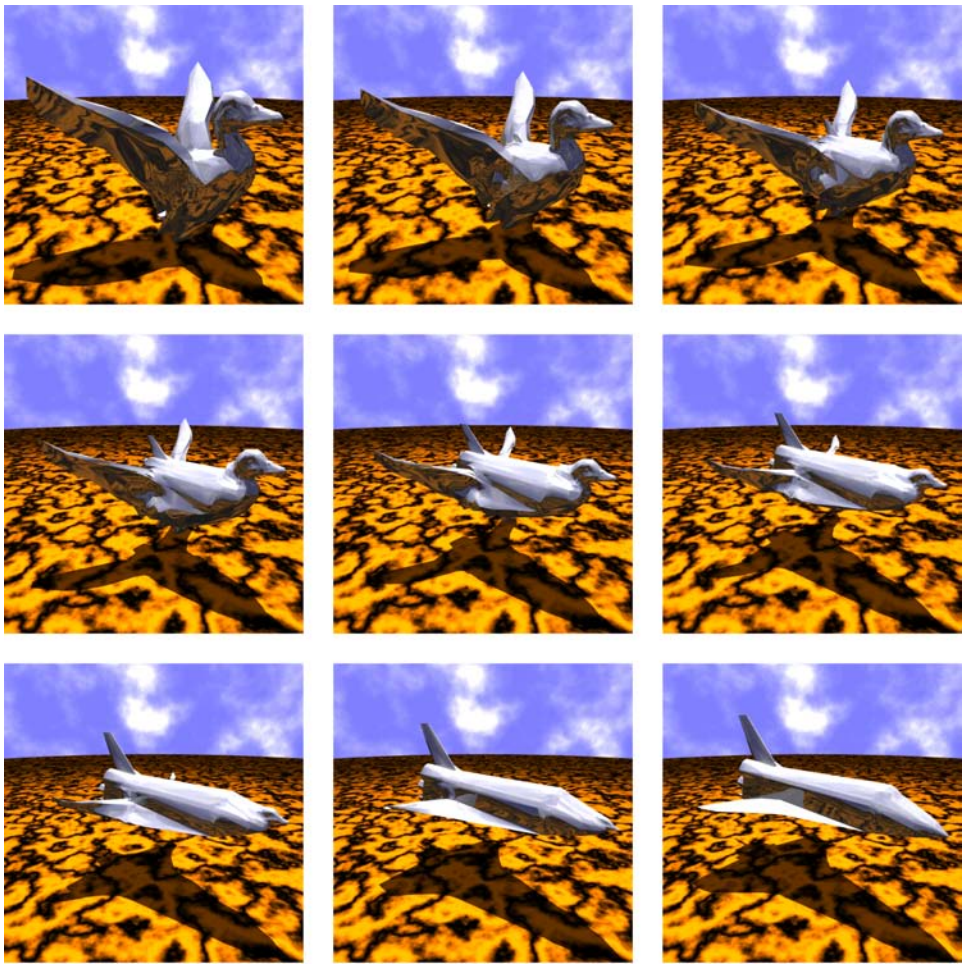


Figure 4.1: A morph sequence obtained by linear interpolation of merged embeddings on the sphere.

Figure 4.2: A morph sequence obtained by linear interpolation using the base domains depicted in Figure 4.3. Reprinted from Kanai et al. [2000].

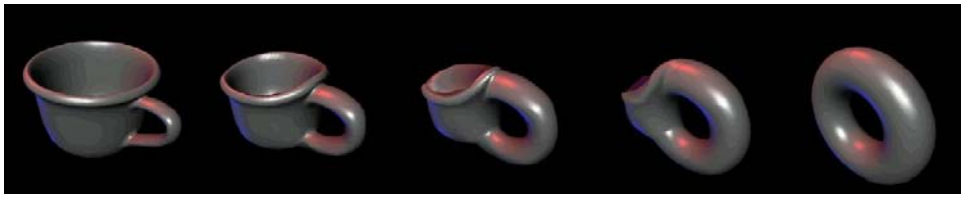


Figure 4.3: A morph of objects with genus higher than zero. Reprinted from Lee et al. [1999].

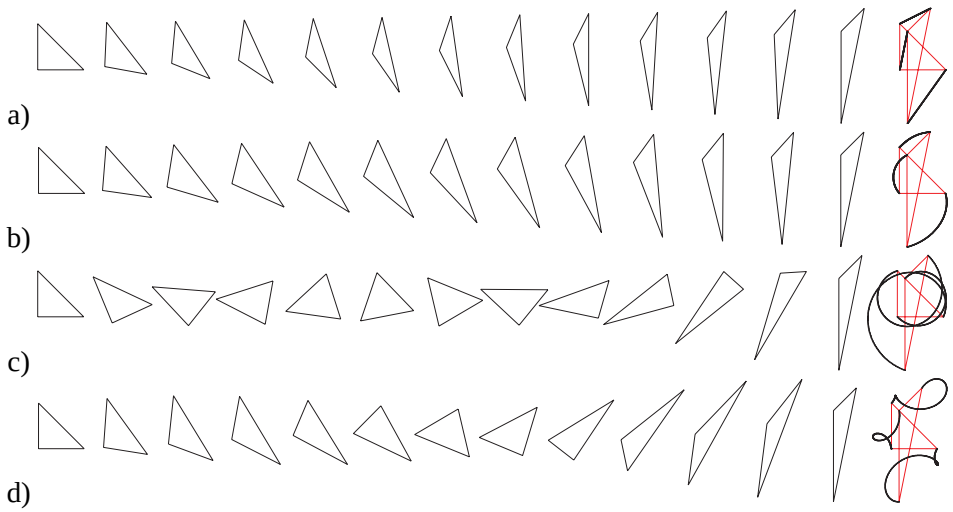


Figure 4.4: Transformations of a single triangle. (a) Linear vertex interpolation. (b-d) An affine map from the source to the target triangle is computed and factored into rotational and scale-shear parts. Intermediate triangles are constructed by linearly interpolating the angle(s) of rotation, the scaling factors, and the shear parameter. (b) is generated using the SVD; (c) shows the results of reducing the overall angle of (b) by subtracting 2π from one of the angles; (d) corresponds to Equation 4.5 and represents the method of our choice. The last column in all rows shows plots of the vertex paths.

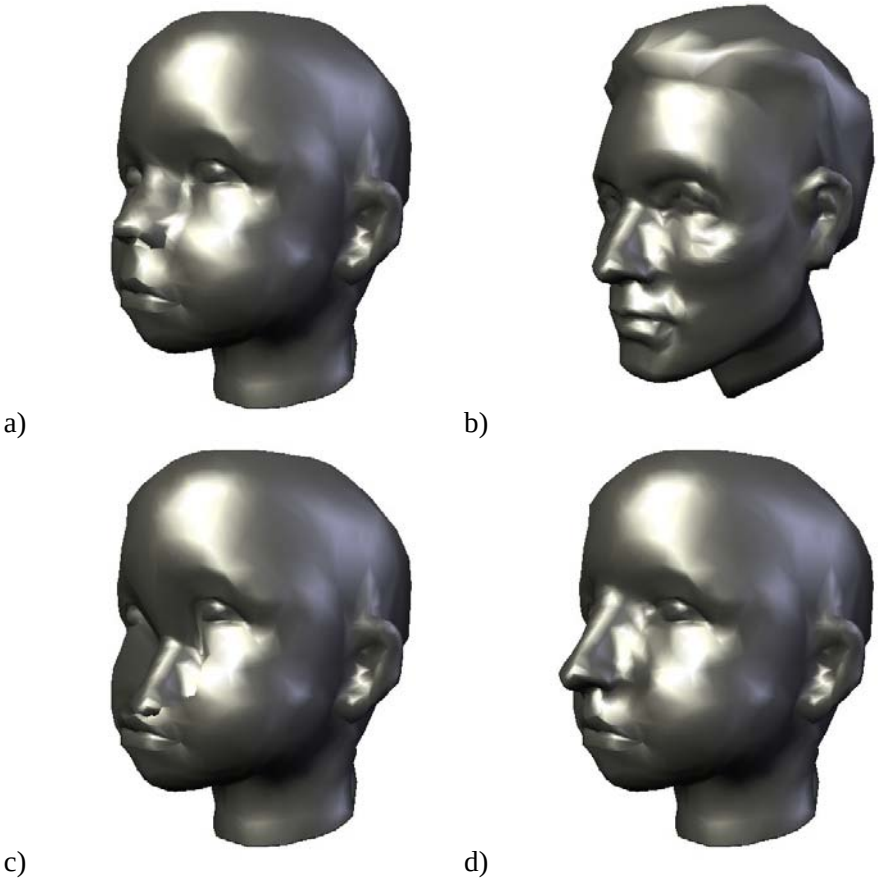


Figure 4.5: Given a mesh with two geometries a) and b) so that corresponding features (eyes, ears, nose, mouth, etc.) are represented by the same vertices in both geometries. If one feature (in this example the nose) is morphed towards the target geometry in absolute coordinates, different positions in space lead to undiserable effects shown in c). The shape in d) shows a more pleasing result achieved by interpolating a differential encoding of the vertices.

4.3.1 Laplacian representation

We have decided to use a rather simple scheme, which is not invariant under rotation, scaling, and shearing. Yet, it is very stable and proved sufficient for the application intended. Let the center of mass of the neighbors of vertex i be

$$\bar{\mathbf{v}}_\mathbf{i} = \frac{1}{|\mathcal{N}(i)|} \sum_i j \in \mathcal{N}(i) v_j \tag{4.16}$$

and let the new representation be the difference of this center of mass to the original position:

$$\tilde{\mathbf{v}}_\mathbf{i} = \mathbf{v}_\mathbf{i} - \bar{\mathbf{v}}_\mathbf{i} \tag{4.17}$$

For an illustration see Figure 4.6. If we write all vertices as a vector the forward

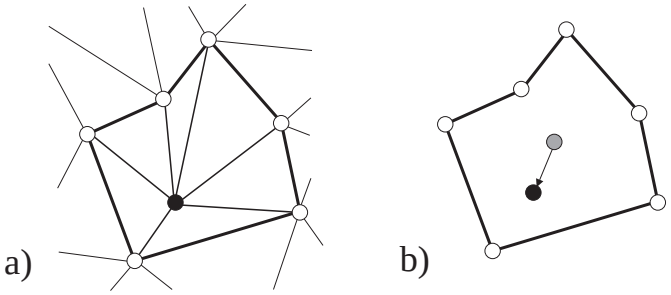


Figure 4.6: A vertex (black) and its neighborhood ring (white) in a). In Laplacian coordinates a vertex is represented by the difference to the centroid of its neighbors (b).

transformation (from absolute to relative coordinates) can be represented in matrix form. Let $A = \{a_{ij}\}$ be the adjacency matrix of the $\mathcal{M} = (K, V)$, i.e.

$$a_{ij} = \begin{cases} 1 & \{i, j\} \in K, \\ 0 & \text{else.} \end{cases}$$

and $D = \{d_{ij}\}$ be a diagonal matrix with $d_{ii} = 1/|\mathcal{N}(i)|$. The transform is represented by $L = I - DA$. Note that L is a Laplacian of the mesh [Taubin 1995]. This is an important observation as it generalizes the approach to shape representations other than meshes, e.g. parametric or implicit functions.

The backward transformation (from relative to absolute coordinates) is, by construction, not unique. It should be uniquely determined up to a translation. This means, $L \in \mathbb{R}^{m \times m}$ should have rank $m - 1$, which is indeed so: Note that DA is a stochastic as well as a normal matrix. A stochastic matrix has an eigenvalue equal 1. In addition, the eigenvectors of normal matrices form a basis of full rank, meaning that all eigenvalues have multiplicity one. It follows that DA has exactly one eigenvalue equal 1 and, thus, L has exactly one eigenvalue equal 0.

The main idea of this work is to morph by linearly interpolating Laplacian coordinates rather than absolute coordinates. Since Laplacian coordinates are linear in absolute coordinates morphing the whole shape (i.e. all vertices have the same transition state) will be the same in absolute and Laplacian coordinates. Yet, if the desired transitions are different for subsets of vertices interpolating Laplacian coordinates yields more reasonable results.

4.3.2 Representing transition states

Given one mesh topology (vertex-edge graph) and d vectors $V_i \in \mathbb{R}^m, i \in \{0, \ldots, d-1\}$ representing different shapes of the mesh we can compute their Laplacian representations $W_i = (I_D A)V_i$. A transition state is a matrix $T \in \mathbb{R}^{m \times d}$. The transition state defines how Laplacian coordinates are combined to form the Laplacian coordinate of the morphed mesh. Each row of T specifies weights for the linear combination of one vertex. If, for example, we want to morph between two meshes a row of T will consist of two entries, which, typically, sum up to one.

The Laplacian representation $\tilde{W}$ with respect to a particular transition state T is given by

$$\tilde{W}(i) = T(i) \begin{pmatrix} W_0^T(i) \\ \vdots \\ W_{d-1}^T(i) \end{pmatrix} \tag{4.18}$$

where $W(i)$ denotes the i-th row of W. The set of respective absolute coordinates is found by solving $(I - DA)\tilde{V} = \tilde{W}$ for $\tilde{V}$.

4.3.3 Computing absolute coordinates

Solving the equation $L\tilde{V} = \tilde{W}$ for $\tilde{V}$ is not possible in a naive way for two reasons. First, L is singular and second, typical meshes will induce matrix dimensions that make the use of explicit techniques prohibitive.

The first problem is easy to overcome. It was already shown in 4.3.1 that the solution $\tilde{V}$ is specified up to a translation. This means fixing one arbitrary vertex will lead to a linear system of equations with full rank.

A practical solution of the resulting linear system should account for the following conditions:

- L is very large and sparse, which prohibits explicit matrix techniques.
- The equation $L\tilde{V} = \tilde{W}$ has to be solved three times, i.e. for the x, y, and z vectors.
- In practice, good approximate solutions are known for $\tilde{V}$ as morphing changes the shape gradually and smoothly from one state to another. Knowing good approximate solutions calls for iterative matrix methods.

- The condition of L is 1. This causes traditional relaxation techniques to converge slowly.

We have decided to use an iterative method with pre-conditioning (see e.g. [Golub & Van Loan 1989]). Since pre-conditioning the matrix equation greatly influences convergence and has to be done only once we have decided to use incomplete LU decomposition as a pre-conditioner. The system is then solved using the conjugate gradient method.

4.3.4 Defining transitions and transition states

A transition state is represented by a matrix T as described in section 4.3.2. A morph, or transition, would be defined as a matrix T changing over time. In practice, one would define several key frames in terms of matrices. Interpolation of key frames is done in matrix space as interpolation in absolute coordinates has to be avoided.

Obviously, it is impossible to specify transition matrices by hand. We either need a user interface or automatic methods.

GUI for defining transition states

The basic requirement for a GUI is to make it easy to define a region of interest (ROI). A ROI is a part of the shape's boundary, for which the transition state will be modified. Modification is performed relative to the current overall transition state of the shape. Technically speaking, prior to the modification the shape is represented by a transition state T.

We have found it very comfortable to specify a ROI by two boundaries. An outer boundary separates the ROI from the rest of the shape. The inner boundary has to lie completely inside the region specified by the outer boundary. The inner boundary specifies the part of the region of interest, which is assigned the user defined values for this ROI. The region between inner and outer boundary is assigned values that range from the user's specification for the inner part to the old transition state. Specifically, we define a distance value d, which is zero inside the inner boundary, one outside the outer boundary, and represents the distance from the boundaries between them. A simple Dijkstra algorithm is used to compute the distances.

Assume the user specifies a new transition state $\tilde{T}$ for the ROI. The new overall transition state T' is defined for each vertex by

$$T' = (1 - d)\tilde{T} + dT \tag{4.19}$$

In a typical modification process, the user selects several ROIs and changes their respective transition states one after the other.

Spatially dependent transition states

Instead of defining the transition state of the shape explicitly using ROIs, the transition state could be defined implicitly. In particular, the transition state might be a function of absolute coordinates. This makes several interesting and important effects possible, e.g. a plane cutting the shape into parts with different transition states.

The definition of a transition state from absolute coordinates is given as a function that maps points in $\mathbb{R}^3$ to weights defining how to combine Laplacian coordinates in this point. It is assumed that the source shapes as well as the generated shape have their centers of mass at the origin of $\mathbb{R}^3$. Since, in the general case, the system of equations to be solved will be non-linear we use a simple heuristic to define a transition state. The absolute coordinates of vertices for each of the source shapes define a transition state. These transition states are averaged to yield the final transition state.

This definition has the advantage that smooth changes of the spatial distribution lead to smooth changes in the shape. For example, assume the user wants to move a plane through a shape so that the two parts correspond to two source shapes. The heuristic given above assures that moving the plane will grow one region and shrink the other, as desired.

Automatic transition sequences

The basic idea is to start the transition at one or several points of the shape and then to "grow" regions representing the target shape until the shape is "covered". In this process, the graph representing the mesh can be exploited.

Nice effects result from "flooding" the mesh, i.e. traversing the graph breadth-first and setting each visited vertex to its target coordinate.

Spatial effects can be achieved as was described in the previous section.

4.3.5 Results and applications

The main application intended for Laplacian coordinates and the presented user interface is, of course, spatially non-uniform mesh morphing. However, other modeling techniques, such as free-from modeling, could also benefit from this representation.

Spatially non-uniform morphing

We have produced several morphable meshes with the techniques described in the previous chapter.

The use of regions of interest for defining transition states and interpolating through these transition states to define an interesting morph sequence is presented in Figure 4.7. A morph sequence from the shape of an egg to the shape of a giraffe is produced. The idea was to let parts of the body pop out the egg one after the

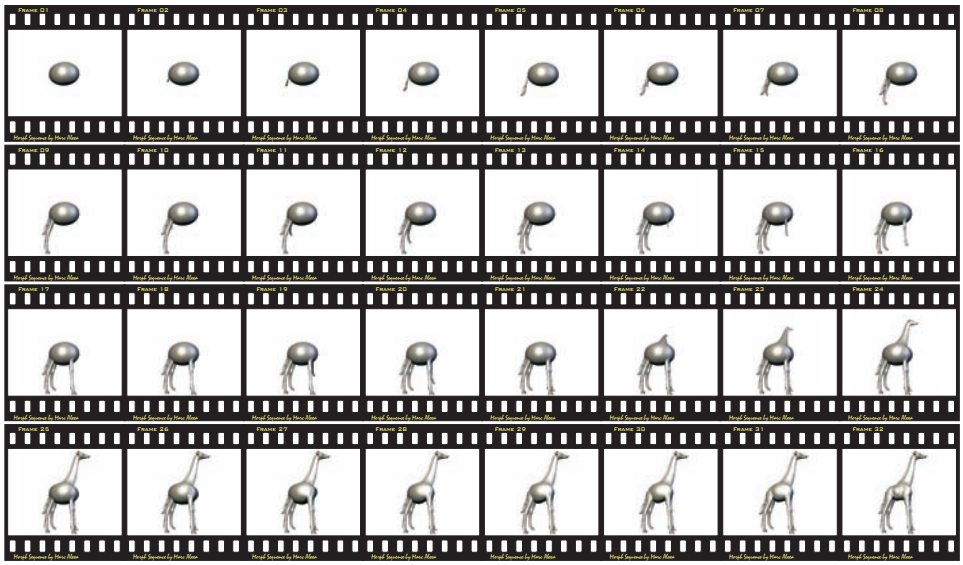


Figure 4.7: A morph sequence between an egg and a model of a giraffe genrated from several transition states. Transition states were defined using regions of interest (ROI). Each ROI corresponds to either the tail, one of the legs, the neck including the head, or the body. Eight transition states were defined, letting the different body parts of the giraffe pop out the egg one after the other.

other. This was achieved by defining ROIs for the different parts of the body and using them to specify transition states. A smooth transition sequence was produce by interpolating the key transition states.

Several examples of defining transition states from absolute coordinates are shown in Figure 4.8. A morphable model of a cow/pig is cut by a plane in two positions. Rather than a sequence several models are presented showing that spatial morph control could be used for modeling. One can imagine how many different creatures could be modeled with virtually no effort using this approach.

An explicit example of modeling using spatial morph control is depicted in Figure 4.9. The intent was to model a Pegasus, i.e. a horse with wings. One easily finds polyhedral models of a horse and an animal with wings. Using ROIs around the wings it is easy to define a transition state which yields the desired result.

In all of these examples morphing is performed between rather different shapes. An interesting application arises from locally morphing among different version of the same shape. In particular, think of different versions representing only parts of the spectrum of the mesh. The eigenvectors of the Laplacian L form a basis, which is the equivalent to a Fourier basis of a mesh [Karni & Gotsman 2000]. This basis could be exploited to define several band-limited versions of the shape. Locally morphing among these shapes has the effect of a local spectral filter.

4.3.6 Free-from modeling

Laplacian coordinates are not only useful for morphing. They can easily be used for free from modeling of meshes by exploiting that they describe the relative position of vertices. The general idea is this:

1. Compute Laplacian coordinates from the original model.
2. Specify the absolute coordinates of several vertices.
3. Compute a linear least squares solution to find the absolute coordinates of the free vertices.

In order to specify fixed vertices we found it convenient to use ROIs, again, defined by an inner and outer boundary. Vertices outside the outer boundary are fixed to their original position. Vertices inside the inner boundary can be moved by the user. The remaining vertices are free and will be computed using the linear least squares approach.

An examples is depicted in Figure 4.10. In the face of a young boy the nose tip is displaced. The remaining vertices are relaxed to approximately fit their Laplacian coordinates using varying sets of free vertices to achieve different effects.

4.3.7 Conclusions

We have presented a well defined method to allow mesh morphing in a spatially non-uniform way. The main idea is to use differential coordinates. Specifically, Laplacian coordinates are introduced, which seem to be well suited for the task of modeling. The advantages of Laplacian coordinates are that they are independent of transformations of the shape and rigorously defined.

However, Laplacian coordinates are sensitive to scaling and rotation of the shape. This can be a problem if corresponding regions of shapes have different size or orientation - despite the fact that shapes are overall reasonably aligned. A representation of coordinates as an affine sum of neighboring vertices would be insensitive to affine transforms. In the future we will try to explore this idea and overcome the problem that neighboring vertices might not be a base of $\mathbb{R}^3$.

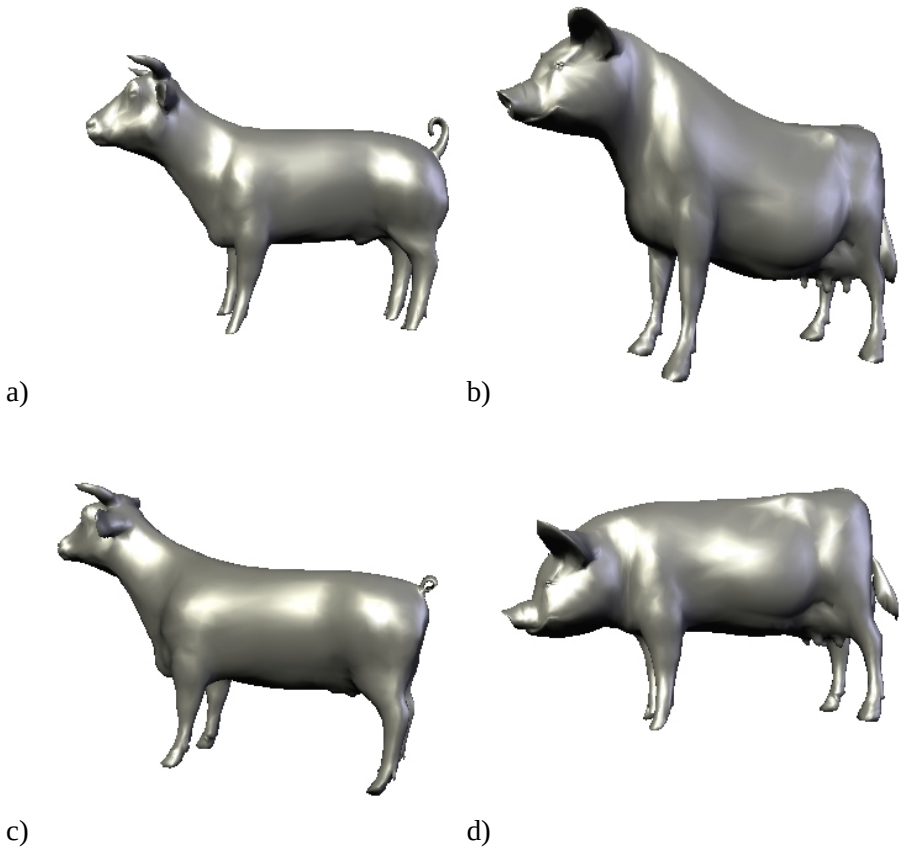


Figure 4.8: Defining the transition state using spatial constraints. Here, a plane is used to split the shape into two parts corresponding to one of two source shapes (representing the models of a cow and a pig). In a) and b) the plane separates head including neck from the body of the animals. In c) and d) the plane is approximately in the middle of the body.

Figure 4.9: Using spatial control for modeling. A Pegasus is modeled by morphing between models comprising its features. Local morph control is used to define a transition state so that only the wings of the duck appear on the horse.

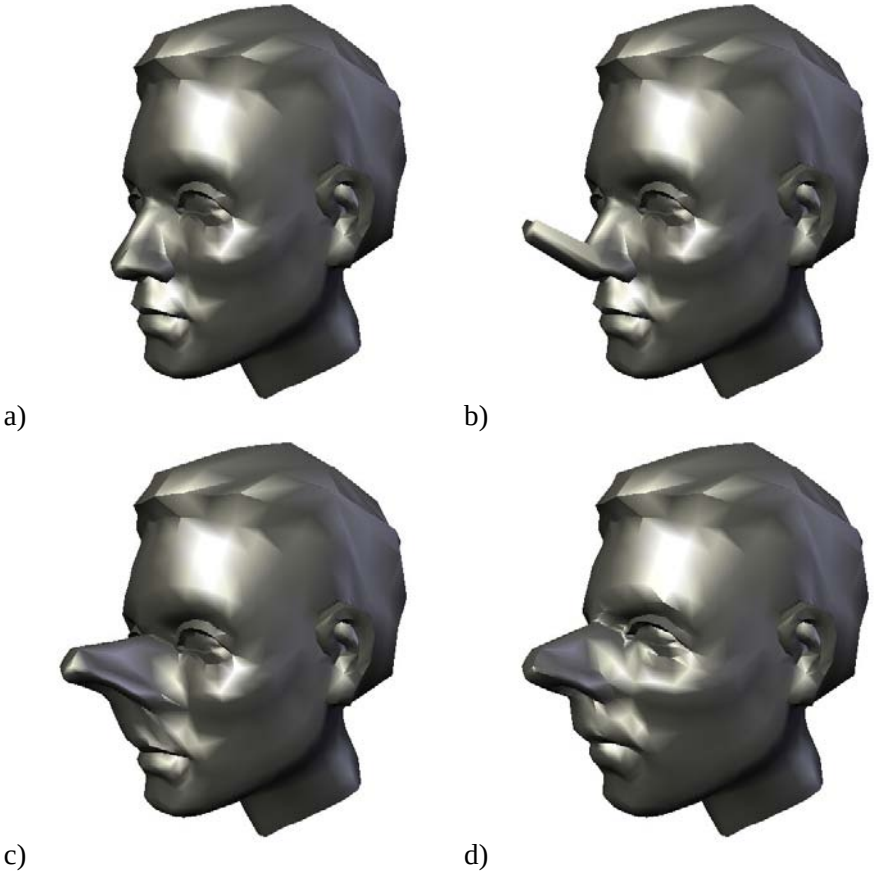


Figure 4.10: Using Laplacian coordinates for free-form modeling. The model of a boy?s face a) is deformed. The nose tip is displaced, b) is showing the displaced vertices. The shapes in c) and d) result from defining a set of free vertices, which are least-square fitted to their Laplacian coordinates.

4.4 Interpolation using isomorphic dissections

Sederberg et al. [1993] introduced techniques that minimize the deformation of the boundaries. Shapira & Rappoport [1995] suggested that a proper morph cannot be expressed merely as a boundary interpolation, but as a smooth blend of the interior of the objects. To achieve such an interior interpolation, they represented the interior of the 2D shapes by compatible skeletons and applied the blend to the parametric description of the skeletons. The automatic creation of corresponding equivalent skeletons of two shapes is involved, and though theoretically possible for all shapes, it seems natural for similar shapes, but ambiguous for rather different shapes like the letters **U** and **T**.

Here, we present an object-space morphing technique that blends the interior of the shapes rather than their boundaries to achieve a sequence of in-between shapes which is locally least-distorting. Assuming that a boundary vertex correspondence of the source and target shapes is given, we apply an algorithm for dissecting the source and target shapes into isomorphic simplicial complexes, i.e. triangles or tetrahedra. Then, we develop a method for interpolating the locations of corresponding vertices, both boundary and interior, along their paths from the source to the target object.

Simplicial complexes allow the local deformation of the shapes to be analyzed and controlled. Floater and Gotsman have used barycentric coordinates to morph compatible triangulations with convex boundary so that no triangles flip on their way from the source to the target configuration [Floater & Gotsman 1999]. However, interpolation of barycentric coordinates is not motivated by or related to physical or esthetic principles.

We start by determining an optimal least-distorting morphing between a source simplex and a target simplex (triangles in the 2D case and tetrahedra in the 3D case). Then, the general idea is to find a transformation which is locally as similar as possible to the optimal transformation between each pair of corresponding simplices.

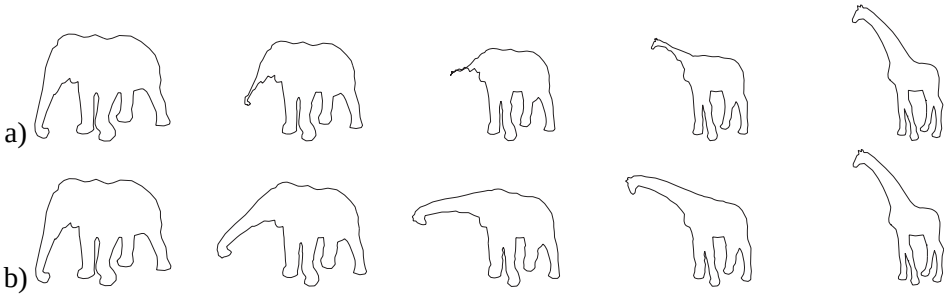


Figure 4.11: Contour blends of the elephant-giraffe example. Simple linear vertex interpolation in (a) vs. as-rigid-as-possible shape interpolation in (b).

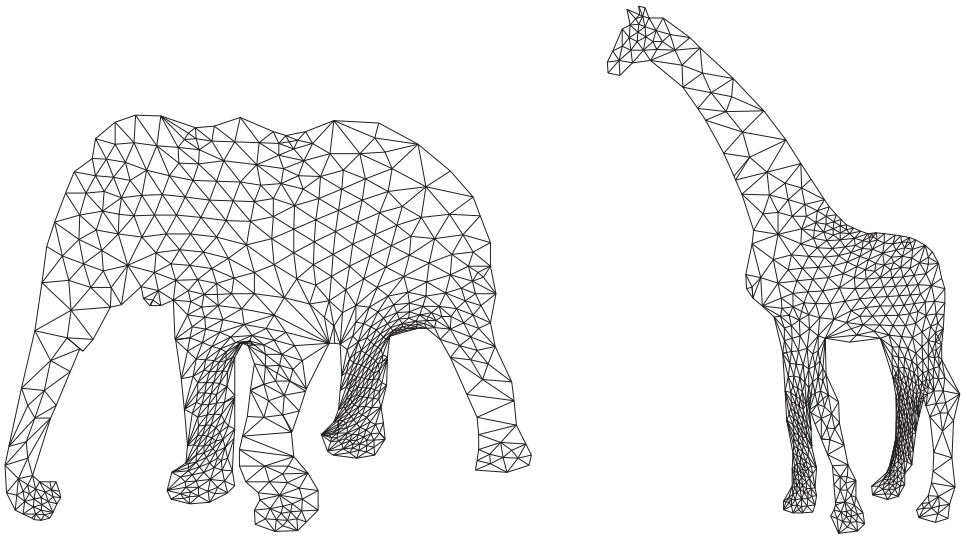


Figure 4.12: The homeomorphic dissections of the shapes in the elephant-giraffe example

4.4.1 Isomorphic dissections of shapes

In this section, we construct isomorphic dissections given two shapes in boundary representation. We assume that the correspondence of the boundaries has been established, i.e. a bijective map between boundary vertices is given. For polygons, reasonable correspondence can be found automatically [Sederberg & Greenwood 1992; Cohen et al. 1997]. In difficult cases, few correspondences could be specified manually and the remaining vertices are matched automatically. For polyhedral objects, several techniques exist, which are based on topological merging introduced by Kent et al. [1992]. Recent work [Gregory et al. 1998; Lee et al. 1999] also allows the specification of corresponding features which seems sufficient to produce acceptable results for a variety of polyhedral models.

Polygons

The problem of constructing a common triangulation for two given polygons is discussed in the literature as *compatible triangulation* [Aronov et al. 1993]. Triangulating a single polygon π is possible using only the vertices of the polygon (e.g. [Chazelle 1990]). However, this is usually not possible for two different polygons. Aronov et al. [1993] show how to triangulate two polygons in a compatible way if at most $O(n^2)$ additional vertices (so-called Steiner points) are allowed. The general scheme [Aronov et al. 1993] is to first triangulate each polygon independently. Then, both polygons are mapped to a regular n-gon so that corresponding boundary vertices coincide. The compatible triangulation is established by overlaying the two edge sets in the convex n-gon. The resulting new interior vertices are then

mapped back into the original polygons, yielding compatible triangulations of the source and target shapes.

We would like to stress that the quality of the blend, in terms of the quality of the in-between shapes, strongly depends on the shape of the simplices. In particular, skinny triangles (or tetrahedra in 3D) cause numerical problems. Thus, in the following, we describe how this scheme can be enhanced to yield compatible triangulations with a significantly better triangle shape.

First, we apply Delaunay triangulations (see any textbook on Computational Geometry, e.g. [de Berg et al. 1997]) as the initial triangulation since Delaunay triangulations maximize the minimum interior angle and, thus, avoid skinny triangles. Of course, any skinny triangle in the independent triangulations is inherited by the merged triangulation. Moreover, Delaunay triangulations are unique, and similar regions in the shapes will result in similar triangulations. Thus, skinny triangles resulting from the overlay process can be avoided.

Nevertheless, the merged triangulations still have skinny triangles, and further enhancement is required to avoid numerical problems. We optimize the triangulations by further maximizing the minimum interior angle, which is known to be a reasonable triangulation quality criterion (see e.g. [de Berg et al. 1997]). We use two independent operations:

1. Moving interior vertices. Freitag et al. [1999] show how to find vertex positions which maximize the minimum angle for a given triangulation.

2. Flipping interior edges simultaneously in both triangulations. This procedure follows the edge flip criteria used in Delaunay triangulation. Given that an edge flip is legal in both triangulations, it is performed if the operation increases the overall minimum angle.

The above two operations are applied in turn until no valid flips are necessary. Convergence is assured since each step can only increase the minimum angle. We call this procedure *compatible mesh smoothing*. The smoothing step optimizes the compatible triangulations without changing the vertex count.

However, we also consider changing the vertex count by means of splitting edges. The split operation is well-defined in terms of topology, if it is applied to both triangulations simultaneously, the isomorphy remains. The idea is to split long edges to avoid long skinny triangles. Splitting edges according to their lengths does not guarantee an increase in triangle quality. In practice, smaller triangles are more likely to be improved by the smoothing step. After each edge split, the triangulations are smoothed. This avoids the generation of edges in regions where the smoothing operation would produce nicely-shaped triangles. Figure 4.14 illustrates the results of splitting edges, as well as of the smoothing process.

4.4.2 Polyhedra

To the best of our knowledge, the three-dimensional analog to compatible triangulations has not been discussed in the literature. Work has been done to dissect

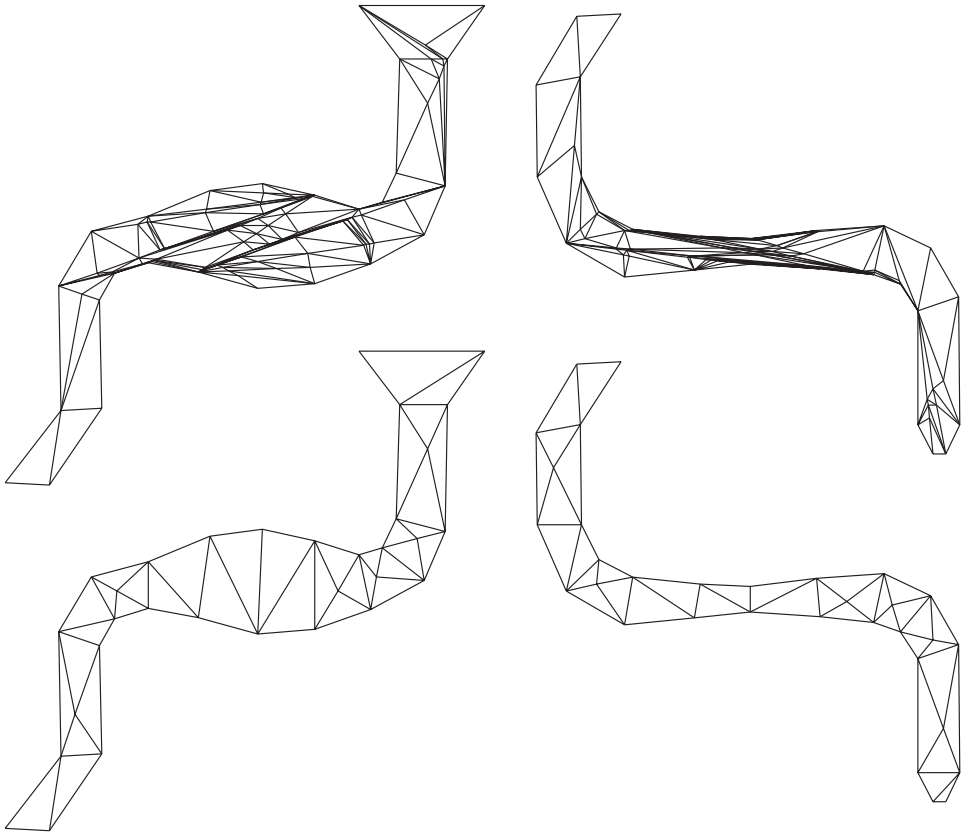


Figure 4.13: A comparison of compatible triangulations. The upper row shows triangulations generated from using ear-capping for the initial triangulation step. Initial triangulations are overlaid on a convex domain to produce compatible triangulations. The triangulations in the lower row were generated with the same general procedure, but using initial Delaunay triangulations. Far fewer triangles are induced, since Delaunay triangulations yield similar partitioning for similar regions.

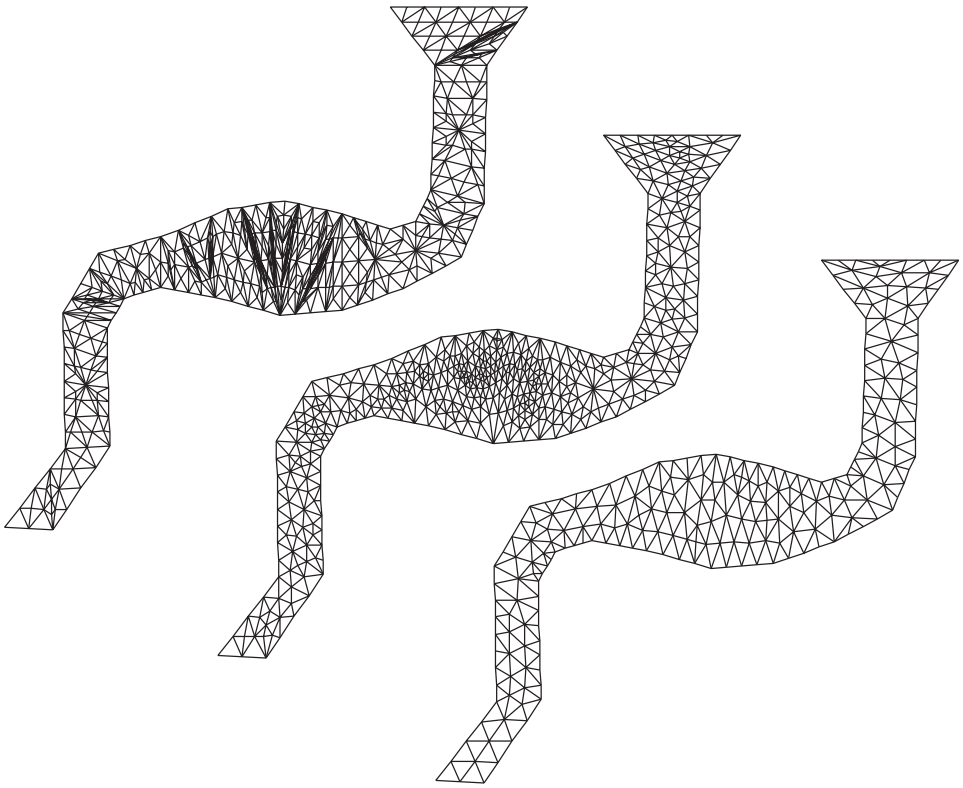


Figure 4.14: The mesh refinement process. In the first row, the merged Delaunay triangulations from Figure 4.13 are refined by edge splits until all edge lengths are bounded. The second row shows the result of compatible mesh smoothing on this triangulation. The third row shows the actual technique, where splitting and smoothing is performed concurrently. Note that the edge length bound is the same in the first and third row.

polyhedra into simplicial complexes, a process referred to as tetrahedralization. However, the work of Aronov et al. [1993] can be extended to genus 0 polyhedra.

First, the source and target polyhedra are tetrahedralized independently using common techniques, e.g., [Chazelle & Palios 1989]. Then, the tetrahedralizations are mapped to a corresponding convex shape. We as well as Shapiro & Tal [1998] describe methods to map an arbitrary genus 0 polyhedron to a convex shape. Since the source and target polyhedra are assumed to have the same vertex-edge topology and the convexification process is deterministic, the polyhedra are mapped to the same convex shape. The interior vertices of their tetrahedralizations are mapped using barycentric coordinates. The fact that vertices are mapped to a convex shape using barycentric coordinates for interior vertices assures that no tetrahedra will be flipped. Then, an overlay of the two tetrahedralizations is computed, where faces are cut against faces, resulting in new edges. Note that the intersection of two tetrahedra results in four-, five-, or six-sided convex shapes, which are easy to

tetrahedralize. The resulting structure is mapped back into original polyhedra. In case the source and target shapes are not genus 0, they have to be cut into genus 0 pieces which are independently treated as explained above.

4.4.3 Transforming shapes

Given two objects together with a set of point-to-point correspondences between user-defined control (anchor) points, one can define an elastic transformation between the objects that exactly satisfies the correspondences. However, to reduce the distortion of the in-between shapes, it is advisable to determine the *rigid* part of the transformation and interpolate it separately from the *elastic* part [Cohen-Or & Carmel 1998; Cohen-Or et al. 1998; Zhang 1996]. The rotational component of the rigid part should be interpolated so that the object is non-deforming, e.g. using quaternion interpolation [Shoemake & Duff 1992]. The rigid-elastic decomposition of the warp function and its particular interpolation are so chosen to minimize the distortion of the intermediate shapes. The rigid part performs the general positional changes, while the fine details are gradually changed by the elastic part.

In many applications, this decomposition does improve the morphing results, though it cannot prevent local distortions in cases of body movements which are more involved as may be found in articulated objects. The underlying assumption in [Cohen-Or & Carmel 1998; Cohen-Or et al. 1998; Zhang 1996] is that the movement can roughly be approximated by rotation, stretching and translation. If we consider objects such as animals' bodies or sophisticated mechanical objects, such as industrial robots, it is clear that even the simplest movements cannot be well approximated by a single rotation and translation. To reduce distortions in transformations of bodies comprising local rotations, the decomposition should be more elaborate. The idea is to determine local non-distorting motions rather than a global one. The composed shape morphing should behave locally as close as possible to the ideal local ones. Figure 4.11 shows a blend between an elephant and a giraffe. The two shapes are aligned and a single rotation cannot prevent the distortions of a linear interpolation, whereas the locally least-distorting interpolation yields a pleasing blend of such articulated objects.

Based on a compatible dissection of the interiors of the shapes (see Figure 4.12), we first define local affine transformations. Each of the local linear maps can be separately decomposed into a rotation and a stretch. Thus, locally, we know how to achieve a non-distorting morph. Then, these local transformations are composed into a global coherent non-distorting transformation, which minimizes the overall local deformation. It should be noted that our transformation is (globally) rigidly reducible; that is, if there is a single rigid transformation that aligns the objects, the morph follows such a path.

We only consider simplicial complexes as dissections of shapes. Specifically, a two-dimensional shape is a *polygon* and its dissection a *triangulation*, and a three-dimensional shape is a *polyhedron* and its dissection a *tetrahedralization*. In the following, we introduce an interpolation technique for determining vertex paths

in shape blending, given a source and a target shape represented by homeomorphic (compatible) triangulations. In Section 4.4.1, we show how to compute such homeomorphic dissections from boundary representations. Note that we describe the concept of determining the vertex paths in two dimensions for clarity; the extension to three or more dimensions is straightforward.

4.4.4 Least-distorting triangle-to-triangle morphing

Suppose the triangulation of the source and target shapes consists of only one triangle each. Let the source vertices be $P = (\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3})$ and the target vertices be $Q = (\mathbf{q_1}, \mathbf{q_2}, \mathbf{q_3})$, where vertices with the same index correspond. An affine mapping represented by matrix A transforms P into Q:

$$A\mathbf{p_i} + \mathbf{l} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \mathbf{p_i} + \begin{pmatrix} l_x \\ l_y \end{pmatrix} = \mathbf{q_i}, \quad i \in \{1, 2, 3\} \tag{4.20}$$

We do not take the translation $\mathbf{l}$ into account for shape interpolation since it does not describe any property of the shape itself except for its placing in the scene. Rather, we want to describe intermediate shapes by varying the rotational and scaling parts comprising A, over time. Note that the coefficients of A are linear in the coordinates of the target shape.

$$\begin{aligned}
a_1 &= \frac{q_{1_x}(p_{2_y}-p_{3_y})+q_{2_x}(p_{3_y}-p_{1_y})+q_{3_x}(p_{1_y}-p_{2_y})}{p_{1_x}p_{2_y}+p_{2_x}p_{3_y}+p_{3_x}p_{1_y}-p_{1_y}p_{2_x}-p_{2_y}p_{3_x}-p_{3_y}p_{1_x}} \\
a_2 &= -\frac{u_{1_x}(p_{2_x}-p_{3_x})+q_{2_x}(p_{3_x}-p_{1_x})+q_{3_x}(p_{1_x}-p_{2_x})}{p_{1_x}p_{2_y}+p_{2_x}p_{3_y}+p_{3_x}p_{1_y}-p_{1_y}p_{2_x}-p_{2_y}p_{3_x}-p_{3_y}p_{1_x}} \\
a_3 &= \frac{u_{1_y}(p_{2_y}-p_{3_y})+q_{2_y}(p_{3_y}-p_{1_y})+q_{3_y}(p_{1_y}-p_{2_y})}{p_{1_x}p_{2_y}+p_{2_x}p_{3_y}+p_{3_x}p_{1_y}-p_{1_y}p_{2_x}-p_{2_y}p_{3_x}-p_{3_y}p_{1_x}} \\
a_4 &= -\frac{u_{1_y}(p_{2_x}-p_{3_x})+q_{2_y}(p_{3_x}-p_{1_x})+q_{3_y}(p_{1_x}-p_{2_x})}{p_{1_x}p_{2_y}+p_{2_x}p_{3_y}+p_{3_x}p_{1_y}-p_{1_y}p_{2_x}-p_{2_y}p_{3_x}-p_{3_y}p_{1_x}}
\end{aligned} \tag{4.21}$$

Intermediate shapes $V(t) = (\mathbf{v_1}(t), \mathbf{v_2}(t), \mathbf{v_3}(t))$ are described as $V(t) = A(t)P$, where $A(t)$ is defined as explained in section 4.1.1.

Note that the rotation of the triangle does not contribute to its shape. However, this is no longer true for more than a single triangle, as we shall see in the next section, which discusses the generalization to more than one triangle.

4.4.5 Closed-form vertex paths for a triangulation

We now consider a triangulation $\mathcal{T} = \{T_{\{i,j,k\}}\}$ rather than a single triangle. Each of the source triangles $P_{\{i,j,k\}} = (\mathbf{p_i}, \mathbf{p_j}, \mathbf{p_k})$ corresponds to a target triangle $Q_{\{i,j,k\}} = (\mathbf{q_i}, \mathbf{q_j}, \mathbf{q_k})$. For each pair of triangles, we compute a mapping $A_{\{i,j,k\}}$, which can be factored by Eq. 4.5 to determine $A_{\{i,j,k\}}(t)$. Since most of the vertices correspond to more than one triangle, a mapping of all vertices could not (in general) be conforming with all the individual ideal transformations $A_{\{i,j,k\}}(t)$.
Let

$$V(t) = (\mathbf{v_1}(t), \ldots, \mathbf{v_n}(t)), t \in [0, 1] \tag{4.22}$$

be the desired paths of the vertices, satisfying

$$\begin{aligned} V(0) &= (\mathbf{p_1}, \ldots, \mathbf{p_n}) \\ V(1) &= (\mathbf{q_1}, \ldots, \mathbf{q_n}). \end{aligned}$$

We define $B_{\{i,j,k\}}(t)$ to be the matrix in the affine transformation from $P_{\{i,j,k\}}$ to $\mathbf{v_i}(t), \mathbf{v_j}(t), \mathbf{v_k}(t)$, i.e.

$$B_{\{i,j,k\}}(t)\mathbf{p_f} + \mathbf{l} = \mathbf{v_f}(t), \quad f \in \{i,j,k\} \tag{4.23}$$

Note that the coefficients of $B_{\{i,j,k\}}(t)$ are linear in $\mathbf{v_i}(t), \mathbf{v_j}(t), \mathbf{v_k}(t)$. We define an intermediate shape $V(t)$ as the vertex configuration which minimizes the quadratic error between the actual matrices $B_{\{i,j,k\}}(t)$ and the desired ones $A_{\{i,j,k\}}(t)$. This quadratic error functional is expressed as

$$E_{V(t)} = \sum_{\{i,j,k\} \in \mathcal{T}} \left\| A_{\{i,j,k\}}(t) - B_{\{i,j,k\}}(t) \right\|^2, \tag{4.24}$$

where $\| \cdot \|$ is the Frobenius norm. In this expression, the $A_{\{i,j,k\}}(t)$ are known for a fixed time t and each $B_{\{i,j,k\}}$ is linear in the $\mathbf{v_i}(t), \mathbf{v_j}(t), \mathbf{v_k}(t)$. Thus, $E_{V(t)}$ is a positive quadratic form in the elements of $V(t)$. In order to have a unique minimizer to $E_{V(t)}$, we should predetermine the location of one vertex, say $v_{1_x}(t), v_{1_y}(t)$, for example, by linear interpolation.
The functional $E_{V(t)}$ can be expressed in matrix form, setting

$$u^T = (1, v_{2_x}(t), v_{2_y}(t), \ldots, v_{n_x}(t), v_{n_y}(t))$$

as

$$E_{V(t)} = u^T \begin{pmatrix} c & G^T \\ G & H \end{pmatrix} u, \tag{4.25}$$

where $c \in \mathbb{R}$ represents the constant, $G \in \mathbb{R}^{2n \times 1}$ the linear, and $H \in \mathbb{R}^{2n \times 2n}$ the mixed and pure quadratic coefficients of the quadratic form $E_{V(t)}$. The minimization problem is solved by setting the gradient $\nabla E_{V(t)}$ over the free variables to zero:

$$H \begin{pmatrix} v_{2_x}(t) \\ v_{2_y}(t) \\ \vdots \end{pmatrix} = -G \tag{4.26}$$

Note that H is independent of t. This means we can invert H and find solutions for time t by computing the corresponding $G(t)$ and a single matrix multiplication:

$$V(t) = -H^{-1}G(t) \tag{4.27}$$

In practice, we compute the LU decomposition of H and find $V(t)$ by back substitution. Furthermore, the computations are separable and are performed independently for the two coordinates. Note that only G depends on the dimension,

Figure 4.15: Transformations of different shapes representing solid objects. Note that parts of the shapes transform rigidly whenever possible. The lowest row shows an example where the shapes have no obvious common skeleton.

while H is the same for the x and y components. Thus, H is effectively of size $n-1 \times n-1$, which means the dominating factor of the computation is independent of the dimension.

The above definition has the following notable properties:

- For a given t, the solution is unique.
- The solution requires only one matrix inversion for a specific source and target shape. Every intermediate shape is found by multiplying the inverted matrix by a vector.
- The vertex path is infinitely smooth, starts exactly in the source shape, and ends exactly in the target shape. These are properties typically difficult to achieve in physically-based simulations.

Figure 4.15 shows transformations of some simple shapes produced with the described method.

4.4.6 Symmetric solutions

While we were satisfied with the degree of symmetry the previously explained approach exhibited in most of our test cases, a symmetric solution can be advantageous – in particular, if the corresponding triangles in the source and target shapes have largely differing area. We can make the solution symmetric by simply blending the optimization problems from both directions. Let $A_f^{\rightarrow}(t)$ be the affine transformation of triangle f from source to intermediate shape at time t, and $A_f^{\leftarrow}(t)$

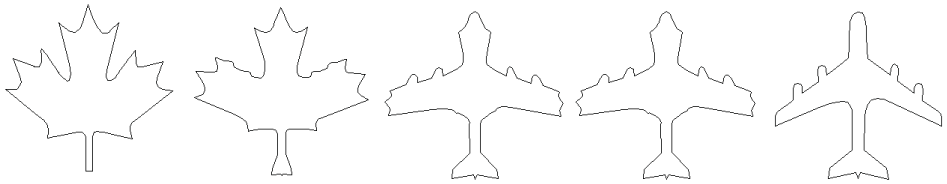


Figure 4.16: The contour of a maple leaf blended with a plane using as-rigid-as-possible shape interpolation. Note that the features of the plane grow out of contour according to the current direction of wings and not their final position.

the respective transformation coming from the target shape. Similarly, we define $B_f^{\rightarrow}(t)$ and $B_f^{\leftarrow}(t)$. We define intermediate $E_{V(t)}$, the vertex configuration at time t, by

$$E_{V(t)} = (1-t)E_{V(t)}^{\rightarrow} + tE_{V(t)}^{\leftarrow} \tag{4.28}$$

where

$$E_{V(t)}^{\rightarrow} = \sum_{f \in \text{ Tri}} \left\| A_f^{\rightarrow}(t) - B_f^{\rightarrow}(t) \right\|^2 \tag{4.29}$$

$$E_{V(t)}^{\leftarrow} = \sum_{f \in \text{ Tri}} \left\| A_f^{\leftarrow}(1-t) - B_f^{\leftarrow}(1-t) \right\|^2 \tag{4.30}$$

With this definition, we lose the advantage of only one matrix inversion for given source and target shapes. Instead, every time t requires the solution of a linear system of equations. Whether the computation times are acceptable depends on the shapes and the desired application.

4.4.7 Results and Conclusion

We have applied the techniques explained above to various inputs. The two-dimensional shapes are generated by extracting a contour out of an image. For the correspondence of contours, we defined manually several vertex-to-vertex correspondences, while the remaining vertices were automatically aligned. The resulting polygons were dissected as described above. In Figures 4.17, 4.18, 4.19, and 4.23, the triangulations were used to map a texture to the shape (as was suggested by Tal & Elber [1999]). Textures were extracted with the contours from the source images. More elaborate techniques for space-time control (e.g. [Lee et al. 1995]) could be easily integrated in our work to give the user more control as to what is transformed and when. Also note that the techniques are not restricted to simple polygons.

Since our technique interpolates shapes "naturally" in the sense that it preserves parts that just change relative position or orientation, it could be also used to extrapolate beyond the source and target shapes. Figure 4.20 demonstrates this with the example of Leonardo DaVinci's studies on proportions (see Figure 4.23

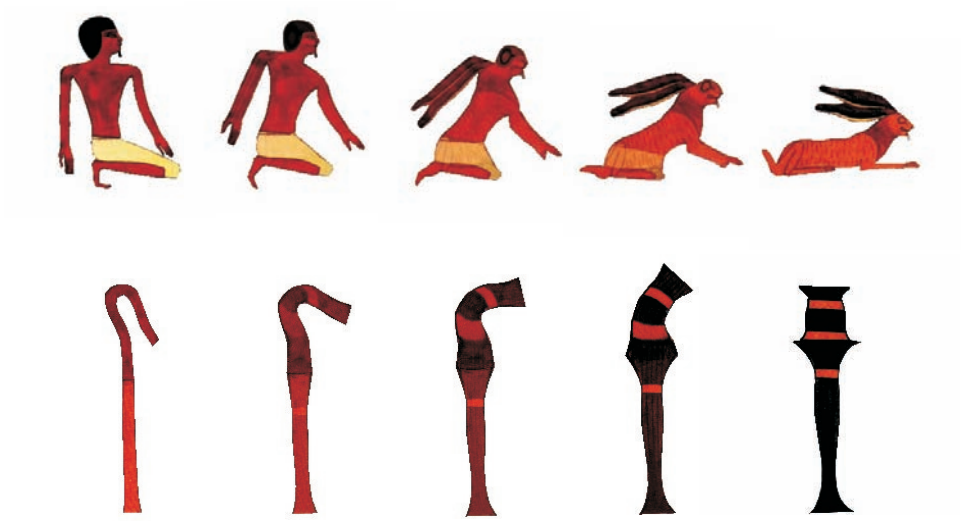


Figure 4.17: Morphs between Egyptian art pieces using textures from the original images. Contours are blended using as-rigid-as-possible shape interpolation and texture colors are linearly interpolated.

Figure 4.18: Contour blend of a penguin and a dolphin using only the texture of the penguin.

Figure 4.19: Morph between photographs of a tiger and a dinosaur.

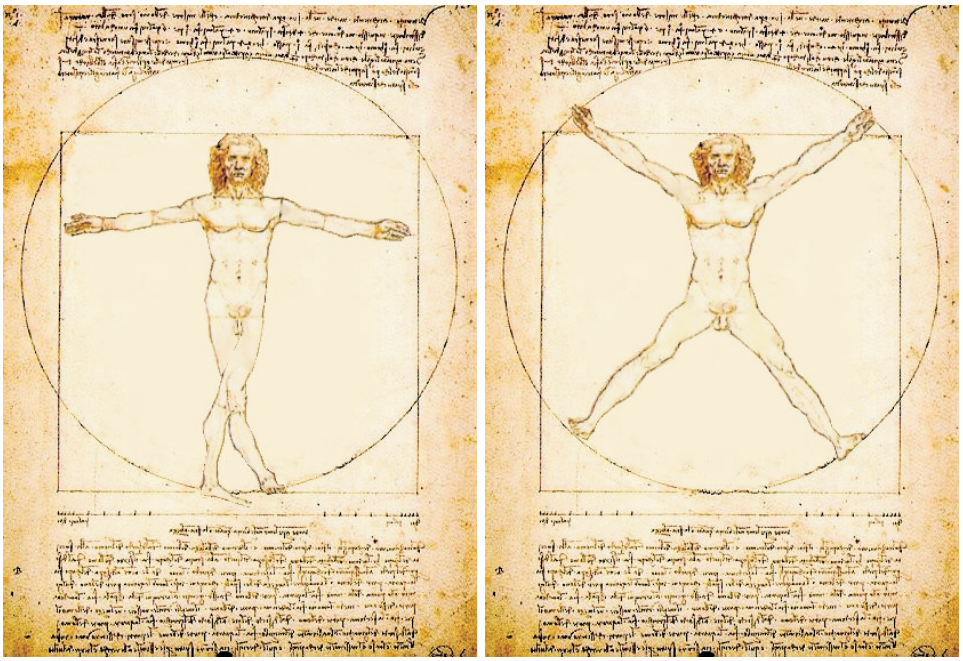


Figure 4.20: Shape extrapolation. Using as-rigid-as-possible shape interpolation, shapes can be naturally extrapolated beyond the source and target shapes. The images show the human figure of Leonardo DaVinci's proportions at time values -0.5 and 1.5.

for the interpolation). We can generate shapes for time values -0.5 and 1.5 while preserving the proportions of the human figure.

We have also applied the interpolation technique to three-dimensional models. The examples in Figure 4.21 were generated by using deformed versions of a polyhedral model. Note the difference between linear vertex interpolation (upper row) and as-rigid-as-possible interpolation (lower row). In Figure 4.22, morphable polyhedral models were generated using topological merging. As in the two-dimensional case, the vertex paths result from defining transformations for each pair of corresponding tetrahedra by factoring the affine transform into rotational and stretching components and, then, minimizing the deviation from these ideal transformations.

The current implementation seems to be robust and fast. The most time-consuming step is optimizing triangle shape. Without optimizing triangle shape numerical problems are likely to occur. In all our examples no simplex changed orientation (i.e. flipped), however, we have not been able to prove this to be a property of our approach.

The examples clearly demonstrate the superior quality of our approach compared to plain linear vertex interpolation. Additionally, it offers the possibility to texture the shapes, so that shape blending becomes applicable to images. In turn,

Figure 4.21: A simple example of three-dimensional objects. The difference of linear and as-rigid-as-possible vertex interpolation is demonstrated on a bent cigar-like shape.

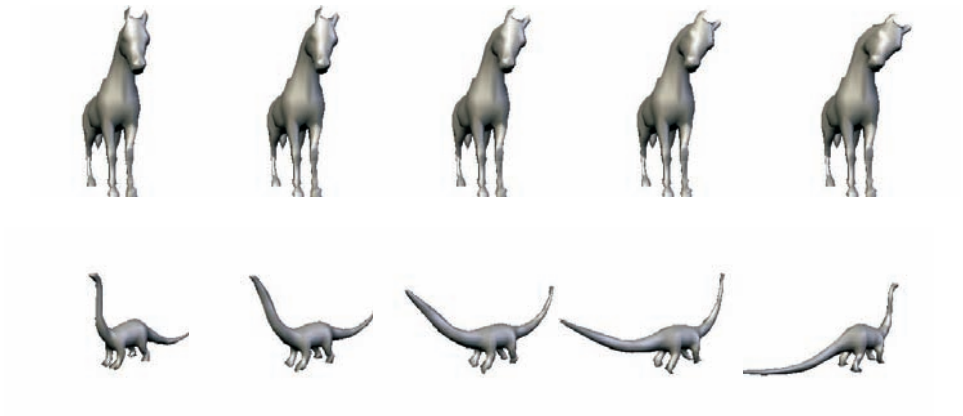


Figure 4.22: Our technique is also useful to mimic motions of articulated three-dimensional objects in case the underlying skeleton is missing, as demonstrated for a horse turning its head. The example in the lower row was produced using a polyhedral morphing technique (facilitating topological merging). Note that the lengths of the tails/necks are preserved.

traditional image morphing techniques could serve to generate the homeomorphic dissections of the shapes and, thus, make use of more advanced vertex/pixel interpolation technique(s). However, the quality of a morph lies in the eye of the beholder. Nevertheless, there is a clear consensus that - lacking other information - the geometry along the morph sequence should change monotonically with no superfluous distortions. The idea of as-rigid-as-possible shape interpolation is to avoid distortions as much as possible and let angles and scales change linearly. We believe that this captures the notion of the above-mentioned consensus.

Despite this, shape blending is always an esthetic problem and no automatic method will meet the needs that arise in different applications. Consequently, user interaction will always be necessary to produce the desired results. Nevertheless, we believe that more elaborate methods for shape blending simplify and minimize the involvement of the designer.

Finally, we want to mention that dissections of shapes seem to extend the concept of skeletons while fully capturing their information. Dissections are more powerful in representing the mechanics of shapes as they allow fine grained analysis of local behavior. In many cases, shapes naturally have no skeleton or their metamorphoses could not be described in terms of a skeleton. These benefits come along with easier and less ambiguous computation of dissections as compared to skeletons.

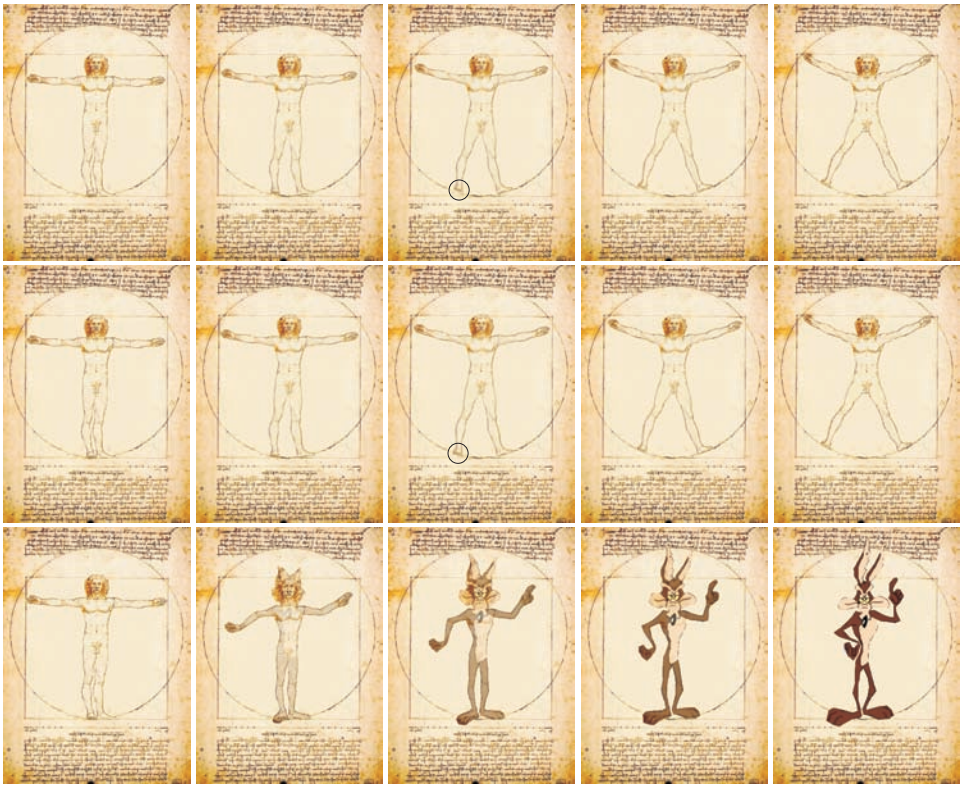
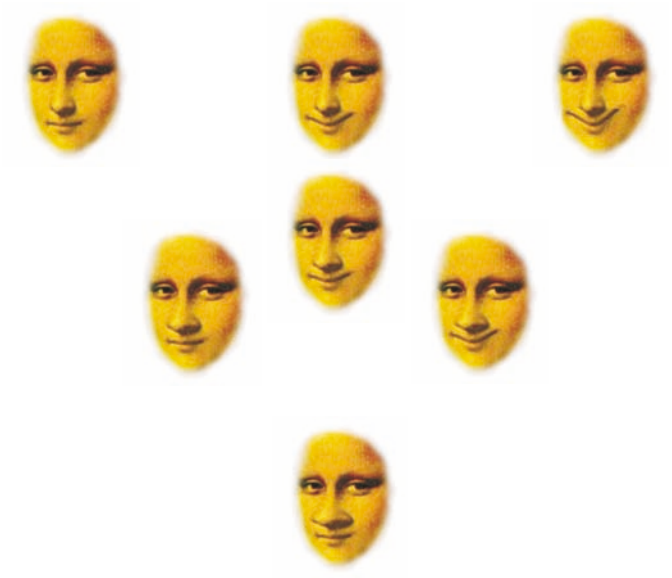


Figure 4.23: Leonardo DaVinci's studies on proportions. The two rows contrast linear and as-rigid-as-possible interpolation of the figure. The difference is subtle, but Leonardo proves our method right (see the feet leaving the circle in the linear interpolation). The last row shows a transformation with more involved movements

Chapter 5

Spaces of shapes from morphing



In this chapter we formalize the concept of spaces from morphing [Alexa & Müller 1998a; Alexa & Müller 1999a]. Therefore we take a closer look at existing morphing techniques in order to identify the core properties of morphing. Based on this view of morphing we will give a definition of the general spaces from morphing.

By defining morphing in an abstract way we will get a scheme to classify existing morphing techniques. Since the properties of morphing spaces will depend on the properties of the morphing techniques, the classification of morphing techniques allows to predict the properties of morphing spaces. Moreover, certain properties of the morphing space will result in more efficient and elegant algorithms for the analysis of objects. This will be shown later.

5.1 Definition of morphing

In order to find a formal definition of morphing we pose ourselves the question: What do we intuitively think of, when we speak about morphing? The answer could read as follows: Given a source and a target object, morphing will produce a transition between these two objects. The answer has a certain strength, i.e. it does not restrict morphing to a certain application. For instance, the use of morphing as an animation tool might be very common, but it does not seem to be a property of morphing itself. On the other hand, morphing cannot be reduced to the production of single objects.

Notice that we already made a first observation: Morphing operates on objects - these can be real or virtual. Morphing needs two objects as input and produces one object as output. All of these objects belong to the same set of objects. We will denote this as follows:

Axiom 5.1 *Morphing operates on an object set.*

In the following we will use the identifier Ω whenever we refer to the object set. The objects of Ω will be identified by upper-case Latin letters. Since Ω is the domain of morphing we assume $\Omega \neq \emptyset$. We want to use Ω to define properties of morphing, thus we need at least an equivalence relation on Ω , which we denote with $=$. Note that we cannot imply that Ω is ordered and we (in general) have no metric on Ω .

We believe there are only two more core properties of morphing, and a third property that is typically demanded for most morphs. These properties concern the objects produced by morphing, which are commonly referred to as in-between objects. The first two properties are covered by the following axiom.

Axiom 5.2 *The set of in-between objects produced by morphing between two objects is ordered and dense.*

We say that morphing induces an ordering on the objects resulting from morphing. The term dense reflects the fact, that for every two in-between objects $A < B$ there exists an in-between object C with $A < C < B$.

The third property is continuity of the in-between objects. For two reasons we do not include continuity in our axioms of morphing. First, there exist useful applications where the morph cannot be continuous (morphing between non homeomorphic objects), and second, without other properties we cannot formalize continuity (for instance, we have no metric on Ω). But nevertheless continuity might be of interest in the application, and after the definition of morphing based on the above axioms we introduce a criterion to assure continuity.

The easiest way to obey the axioms is to represent the in-between objects by an ordered and dense set. The canonical choice is the interval of real numbers [0,1]. We can look at the reals of this interval as one parameter that - together with a source and target object - identifies an in-between object. We call this parameter the *transition parameter*. Thus morphing can be described as a function of three

variables. Choosing the transition parameter 0 will reproduce the source object and conversely the transition parameter 1 will reproduce the target object. Formally:

Definition 5.1 *Let $A, B \in \Omega$ and $t \in [0,1]$. We will refer to $m : \Omega \times \Omega \times \mathbb{R} \mapsto \Omega$ as the morphing function, if m is well-defined for all $t \in [0,1]$ and the following equations hold:*

$$m(A, B, 0) = A \tag{5.1}$$
$$m(A, B, 1) = B \tag{5.2}$$

In most of the following discussion it will be more convenient to think of m as a generalized morphing function. That means the transition parameter of m is not limited to the interval $[0,1]$ but can be assigned any real value. The above definition remains the same, except that [0,1] is replaced by the real numbers $\mathbb{R}$ (the interval $[-\infty, \infty]$). The transition parameters 0 and 1 will still reproduce the source and the target object, respectively. Transition parameters outside of $[0,1]$ can be understood as an extrapolation of the objects.

5.2 Properties of morphing functions

We will define some common mathematical terms for morphing functions. We limit the discussion to those necessary for the properties of morphing spaces.

Definition 5.2 *A morphing function is* injective*, if for all $A, B \in \Omega$ the following implication holds:*

$$t \neq t' \Rightarrow m(A, B, t) \neq m(A, B, t') \tag{5.3}$$

We need the property of injectivity only for the transition parameter. Note that we can invert the morphing function on its domain exactly when it is injective. We will denote the inverse of the morphing function with w.r.t. to the transition parameter as $w : \Omega \times \Omega \times \Omega \mapsto t$. The following is the characteristic property of w:

$$w(A, B, m(A, B, t)) = t \tag{5.4}$$

For the next property we need to introduce a formal description concerning the set of in-between objects:

$$\Psi_{AB}(a, b) = \{X \in \Omega | X = m(A, B, t) \vee t \in [a, b]\} \tag{5.5}$$

Now we formalize the following idea: Suppose we produce an object C by morphing between objects A and B. A morph between A and C or C and B will result in objects that could have been produced by morphing between A and B, as well.

Definition 5.3 *A morphing function is* compositionable*, if the following equations hold for every $C = m(A, B, \tau)$:*

$$\Psi_{AC}(0, 1) = \Psi_{AB}(0, \tau) \tag{5.6}$$
$$\Psi_{CB}(0, 1) = \Psi_{AB}(\tau, 1) \tag{5.7}$$

While compositionable morphing functions are crucial for the construction of morphing spaces they also allow the discussion of continuity. In many applications a continuous morph seems desirable. The axioms and the resulting definition of the morphing function did not suffice to define continuity in the common way, since we have no metric on Ω . We have found to a way to describe continuous morphing functions by an analogy to theorems from calculus.

Theorem 5.1 *A compositionable morphing function* m *is* continuous*, if every* $C = m(A, B, \tau)$ *can be represented by a composition of functions* $m(A, B, \chi$ *with every fixed* $\chi, 0 < \chi < 1$.

In order to prove the above theorem we show the analogy to the Bolzano theorem of real calculus. Calculate $D_1 = m(A, B, \chi)$. If $D_1 = C$ we are done. If not, identify the interval $\Psi_{AD_1}(0, 1) = \Psi_{AC}(0, \chi)$ or $\Psi_{D_1B}(0, 1) = \Psi_{AC}(\chi, 1)$ to which C belongs (remember that m is compositionable). According to the chosen interval calculate $D_2 = m(A, D_1, \chi)$ or $D_2 = m(D_1, B, \chi)$. Again if $D_2 = C$ we are done and if not the interval will be divided by the calculation of D_3. Eventually C is either represented by a D_i or by an infinite process of interval divisions. The latter representation is analogous to the Bolzano's theorem, which requires continuity of the underlying set. More specifically it is the representation of a value by interval division that needs the property of continuity.

5.3 Definition of morphing space

Suppose we are given n base objects that are the basis of our morphing space. Intuitively, the morphing space consists of all objects that can be produced by applying the morphing function to the base objects and to objects that have been produced by applying the morphing function to the base objects, and so forth. Formally, we use the following definition:

Definition 5.4 *Given an object set* Ω *,* base *objects* $B_0, B_1, \ldots, B_{n-1} \in \Omega$*, and a morphing function* m*: Let the sets* $\Phi \subseteq \Omega$ *be defined by:*

$$\Phi_0 = \{B_0, B_1, \ldots, B_{n-1}\} \tag{5.8}$$

$$\Phi_i = \{m(C, D, t) | C, D \in \Phi_{i-1}, i \geq 1\} \tag{5.9}$$

The set $(\Phi = \Phi_\infty) \subseteq \Omega$ *is called the* general morphing space.

This definition expresses the idea of the space of all objects generated by morphing between the base objects but is not really helpful in practice. Since we have n base objects we are interested in an description (or representation) of elements relative to these objects but not to the actual calculation. We want to describe objects in terms of weights representing the shares of each of the base objects: Every element $A \in \Phi$ has a representation $a \in \mathbb{R}^n$ with vector elements a_i representing the share of base object B_i. In order to analyze this case we restrict the defintion of general morphing spaces as follows:

Definition 5.5 *Given an object set* Ω *,* base *objects* $B_0, B_1, \ldots, B_{n-1} \in \Omega$*, and a morphing function* m*: Let the sets* $\Phi \subseteq \Omega$ *be defined by:*

$$\Phi_0 = B_0 \tag{5.10}$$

$$\Phi_i = \{m(A, B_i, t) | A \in \Phi_{i-1}, i \geq 1\} \tag{5.11}$$

The set $\Phi_n \subseteq \Omega$ is called the finite dimensional morphing space.

Obviously, we can represent the elements of Φ_n uniquely by vectors in $\mathbb{R}^n$. This might be useful in some special applications but in general this will result in two major drawbacks:

1. The space is limited by the fixed order of base objects, i.e. the space does not contain all objects that might be produced by morphing among the base objects.

2. The space is not closed against the morphing function, i.e. morphing between two elements of the space can result in a non-member of the space (this is due to the limited dimension).

Therefore, we strive to answer the following question: What conditions must m satisfy such that $\Phi_n = \Phi_\infty$ for every set of base objects? Note that the combination of the following two conditions seems to be sufficient:

1. The morphing function must be order independent when applied to more than two objects.

2. The morphing function must be compositionable.

Since we have a condition for the second requirement, we only have to find a condition for the first. In order to present a sufficient criterion we first introduce linear morphing functions.

Definition 5.6 *A morphing function is* linear *if for all* $A, B \in \Omega$ *the following equation holds:*

$$m(m(A, B, x), m(A, B, y), z) = m(A, B, x + z(y - x)). \tag{5.12}$$

Interestingly, linearity encompasses composition:

Theorem 5.2 *Linear morphing functions are compositionable.*

In contradiction to the statement we assume the linear morphing function m is not compositionable. We investigate an element $C = m(A, B, \tau)$ and the sets $\Psi_{AC}(0, 1)$ and $\Psi_{CB}(0, 1)$. Since m is not compositionable at least one of the sets is not equal the corresponding sets $\Psi_{AB}(0, \tau)$ and $\Psi_{AB}(\tau, 1)$. But the bijection

$$m(A, m(A, B, \tau), z) = m(A, B, \tau z)$$

connects the elements of the first intervals and the bijection

$$m(m(A, B, \tau), B, z) = m(A, B, \tau + z - \tau z)$$

connects the elements of the second intervals.

However, linearity is still not enough to ensure order independence. For that we also need distributivity.

Definition 5.7 *A morphing function is* distributive *if for all $A, B \in \Omega$ the following equation holds:*

$$m(m(A, B, x), m(A, C, x), z) = m(A, m(B, C, z), x) \tag{5.13}$$

We have found a single equation that enforces both properties linearity and distributivity:

$$m(m(A, B, x), m(A, B, y), z) = m\left(A, n\left(B, C, \frac{yz}{x + z(y - x)}\right), x + z(y - x)\right) \tag{5.14}$$

We call morphing functions that satisfy the above equation *neat*. These morphing functions fulfill all requirements for $\Phi_n = \Phi_\infty$. Furthermore, the linearity of neat morphing functions gives rise to the assumption that neat morphing functions make Φ_n an affine space, or, if we pick a zero element a vector space. We proceed with proving this assumption, since the linearity of the space also implies the order independence of the base objects. We will discuss later on what existing morphing techniques could be described in terms of a neat morphing function.

5.4 Morphing space as a affine/vector space

We believe that Φ_n is an affine space, if the morphing function is neat. We found it advantageous to prove this by picking a zero element in the space and to prove that Φ_n together with this zero element is a vector space.

A vector space (or linear space) is characterized by its domain with an addition and a scalar multiplication. In order to show that the morphing space can be a linear space we need a definition for these two operations. We pick the base element B_0 as zero element of the space and denote it with 0. Now we can define the scalar multiplication.

Definition 5.8 *A scalar multiple $\mathbb{R} \times \Phi \mapsto \Phi$ of an element $A \in \Phi$ is given by*

$$\lambda A = m(0, A, \lambda) \tag{5.15}$$

for $\lambda \in \mathbb{R}$.

The definition of an addition is inspired by the graphical addition of vectors:

Definition 5.9 *The addition $\Phi \times \Phi \mapsto \Phi$ of two elements $A, B \in \Phi$ is given by:*

$$A + B = m\left(0, m\left(A, B, \frac{1}{2}\right), 2\right) \tag{5.16}$$

Note here that we are making use of a more general understanding of a morphing function with a transition parameter $t \in \mathbb{R}$. Now one can show, that the morphing space with the above operations is a vector space, if the morphing function is neat. The proof consists of simple equivalence transformations and is therefore omitted.

5.4.1 Representation of elements and dimension of Φ_n

We look at an element of the morphing space as a kind of compound of the base elements. The mathematical way to produce such a mixture is a weighted sum. In our case the weights are the representation of an element. Such, we can write an element $A \in \Phi_n$ as

$$A = \sum_{i=0}^{n-1} x_i B_i \tag{5.17}$$

Consequently, A is completely represented by the x_i. As Φ_n is an affine space we know that

$$\sum_{i=0}^{n-1} x_i = 1 \tag{5.18}$$

The x_i are called barycentric coordinates of A. Because we can choose only $n-1$ of the x_i independently, the dimension of Φ_n is bounded by $n-1$. If we use Φ_n as a vector space with B_0 as zero element, we will actually use the representation $\mathbf{x} = (x_1, \ldots, x_n - 1)$, i.e. dismissing the weight x_0. This has the notational advantage that the null-vector represents B_0. In the remainder of the discussion we will look at Φ_n as a vector space, because it is of greater relevance to the applications than an affine space. Nevertheless, in most cases the statements for affine spaces can be found by simply exchanging "linear" for "affine".

With the definition of scalar multiplication and addition we have readily given an algorithm that constructs an element out of its vector representation. This algorithm is derived from the representation of the elements as a weighted sum. We can also think of this construction process as a map from $\mathbb{R}^{n-1}$ to Φ_n. Note that this map is at least surjective, because every element of Φ_n has a representation in $\mathbb{R}^{n-1}$. Naturally the question arises whether the map is also injective, which would make Φ_n and $\mathbb{R}^{n-1}$ isomorphic.

5.4.2 Isomorphism between Φ_n and $\mathbb{R}^{n-1}$

To answer this question, we need to further investigate our base elements. As in all vector spaces we can speak of linear dependence and independence of objects according to the following definition:

Definition 5.10 *Given elements $B_0, B_1, \ldots, B_{n-1} \in \Omega$. The elements are* linear independent*, if the equation $\lambda_1 B_1 + \ldots + \lambda_{n-1} B_{n-1} = B_0$ has only the trivial solution $\lambda_1 = \ldots = \lambda_{n-1} = 0$. Otherwise the elements are* linear dependent*.*

With this definition we can prove a possible isomorphism between morphing spaces and $\mathbb{R}^{n-1}$. In addition to be neat the morphing function has to be injective. We will show that the linear independence of the base elements and an injective morphing function are both necessary and together sufficient for a morphing space to be isomorphic to $\mathbb{R}^{n-1}$ (given that the morphing space is a vector space according to the above definitions).

To show that the morphing function has to be injective we take a look at one of the axes of Φ_n. Its domain is given by the values of $m(0, B_i, x_i)$. These values are represented by vectors $(\ldots, x_i, \ldots)$. If m is not injective, there exists an element $A = m(0, B_i, \tau) = m(0, B_i, \tau')$ with $\tau \neq \tau'$ having the different representations $(\ldots, \tau, \ldots)$ and $(\ldots, \tau', \ldots)$ and the map from $\mathbb{R}^{n-1}$ to Φ_n would not be injective. Thus, m has to be injective.

If the base elements are not linear independent, there exists a base element that can be represented as a weighted sum of the remaining base elements, e.g.

$$B_0 = \lambda_1 B_1 + \ldots + \lambda_{n-1} B_{n-1} \tag{5.19}$$

so that not $\lambda_i = 0$ and thus, the left-hand side $(1, 0, 0, \ldots)$ and the right-hand side $(1 - \sum_i \lambda_i, \lambda_1, \ldots)$ are not equal. Again the map from $\mathbb{R}^{n-1}$ to Φ_n would not be injective and, therefore, the base elements have to be linear independent.

To show that both conditions are together sufficient we look at an element

$$A = x_1 B_1 + \ldots + x_{n-1} B_{n-1} \tag{5.20}$$

and assume in contradiction to the statement, there would exist another representation

$$A = x'_1 B_1 + \ldots + x'_{n-1} B_{n-1} \tag{5.21}$$

But then we have

$$x_1 B_1 + \ldots + x_{n-1} B_{n-1} = x'_1 B_1 + \ldots + x'_{n-1} B_{n-1} \tag{5.22}$$

and because of the injectivity of m all $x_i B_i, x'_i B_i$ are uniquely determined. The uniqueness permits term-transformations and we get

$$0 = (x'_1 - x_1) B_1 + \ldots + (x'_{n-1} - x_{n-1}) B_{n-1} \tag{5.23}$$

But because the basis $B_0, \ldots, B_{n-1}$ is linear independent the above equation has only the trivial solution

$$x'_1 - x_1 = \ldots = x'_{n-1} - x_{n-1} = 0. \tag{5.24}$$

Thus all representations of A are identical, or in other words the map from $\mathbb{R}^{n-1}$ to Φ_n is injective.

5.5 Algorithms for object synthesis and analysis

The two basic operations in morphing spaces are synthesis and analysis of elements. Synthesis denotes the process of constructing an element in a given morphing space out of a given vector representation. Analysis is the calculation of a vector representation for a given element in a given morphing space.

As will be seen shortly, both problems can be solved without any knowledge of the objects and the morphing technique. The algorithms assume only that a classical morphing technique (i.e. one operating on two objects) is available. In the procedure it is used as a black box: A source and target object and a transition parameter are given as input and an in-between object is returned as output.

5.5.1 Synthesis of objects

In order to find a synthesis algorithm we will use the vector space properties of morphing spaces. With this presumption we can present an algorithm that constructs an element in a proveably optimal way, assuming all we have is a classical morphing technique. We will discuss the case of synthesis in non-linear morphing spaces later.

Assume we want to synthesize an object A with the given vector representation $\mathbf{x}$. The algorithm can be described as follows:

1. Derive A by morphing between the null object B_0 and the projection of A into the $n-2$-dimensional subspace spanned by the base objects without B_0.

2. To calculate the projection, identify a null object in the subspace (i.e. the base object with the least index) and repeat the procedure of steps one and two.

3. If the subspace is one-dimensional (this must happen eventually) calculate the projection by applying the morphing function directly.

4. Calculate the projection in the next higher-dimensional subspace, and so on, until the object is synthesized.

The main problem is to find the vector representation of the projection in the subspace. But as it will turn out, all linear equations have ad hoc solutions and their is no need to solve any matrix equations.

Object A has the representation $(x_1, \ldots, x_{n-1})$ in the original morphing space. We want to find its representation in the space with basis $B_1, \ldots, B_{n-1}$. Therefore, we have to find the intersection of this subspace and a line through B_0 and A. Thus we have

$$\lambda A = B_1 + \mu_1(B_2 - B_1) + \ldots + \mu_{n-2}(B_{n-1} - B_1) \tag{5.25}$$

or in vector formulation

$$\lambda \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \mu_1 \begin{pmatrix} -1 \\ 1 \\ \vdots \\ 0 \end{pmatrix} + \ldots + \mu_{n-1} \begin{pmatrix} -1 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \tag{5.26}$$

The resulting system of linear equations

$$\begin{pmatrix} x_1 & 1 & \ldots & 1 \\ x_2 & -1 & \ldots & 0 \\ \vdots & & \ddots & \vdots \\ x_{n-1} & 0 & \ldots & -1 \end{pmatrix} \begin{pmatrix} \lambda \\ \mu_1 \\ \vdots \\ \mu_{n-2} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{5.27}$$

can be solved easily. By adding all rows we obtain

$$\lambda \sum_i x_i = 1 \quad \Rightarrow \quad \lambda = \frac{1}{\sum_i x_i} \tag{5.28}$$

and by applying this to the j-th row of our linear equations we find

$$\frac{1}{\sum_i x_i} x_j - \mu_{j-1} = 0 \tag{5.29}$$

and, thus,

$$\mu_j = \frac{x_{j+1}}{\sum_i x_i} \tag{5.30}$$

The weights μ_j give the representation for A in the subspace with null element $(1, 0, \ldots, 0)$.

With repeated insertion one eventually obtains the following closed form of the procedure:

$$C_k = m\left(B_k, C_{k+1}, \frac{\sum_{i=k+1}^{n-1} x_i}{\sum_{i=k}^{n-1} x_i}\right), k < n-1 \tag{5.31}$$

The calculation begins with C_{n-1}, because in this case the we immediately obtain $C_{n-1} = B_{n-1}$. Then, $n-1$ applications of the morphing procedure are necessary to find $C_0 = A$.

Now it is already clear, why this algorithm is optimal: We have to combine n objects with a given operation. Clearly, we have to apply this operation at least $n-1$ times. The algorithm has another advantage. If A is really an "in-between" element of the base elements, i.e. we have $x_i \in [0, 1]$ for all i, then all intermediate elements C_k are also "in-between" objects. If we look for example at the definition of addition in morphing spaces, this property is not evident.

In the derivation of the algorithm we actually used the vector space properties of the morphing space. The resulting algorithm is applicable also to non-linear morphing spaces. The only problem in the unique construction of an object is

the ordering of the base objects (since this is the only degree of freedom in the algorithm). If the morphing space is a vector space the input order of the base objects is not relevant in the construction process. It might be the case, that this order is relevant for some morphing techniques. Then, the ordering of the base objects has to remain constant in all applications in order to have reproducible results. In other words, in this case the ordering of the base objects is one parameter of the morphing space.

5.5.2 Analysis of objects

The other important task in a morphing space is to find a description of the object in the space, which represents the object's properties best. Until now, we did not use any additional information to describe the morphing space properties and the synthesis algorithm. The analysis of objects, however, cannot be performed without the definition of a distance function on the objects. Note that the notion of morphing functions actually defines a metric. Yet, we cannot compute it.

For this reason we have to resort to practical solutions for a metric. For example, on images one could use the squared distances of pixel colors in a color space.

For the analysis of objects we cannot use one scheme for linear and non-linear morphing spaces. If the morphing space has the characteristic of a vector space we can use a much more sophisticated algorithm to find the representation. In non-linear morphing spaces we will further distinguish some situations.

In all cases we need to minimize along (one-dimensional) lines. In all cases we have to do this, we know that the minimum is a unique local minimum. We therefore employ a numerical method known as Brent's method [Brent 1973]. This method switches between a golden section search and the approximation of the minimum by a parabola, according to the appropriateness.

Note that this one-dimensional minimization could be seen as a practical solution for finding the inverse element of m. Therefore we will use the symbol w as in the above mathematical discussion.

5.5.3 Analysis in linear morphing spaces

We have found a considerably fast method to analyze an object in a morphing space, if the morphing space is linear and the distance function has a certain property. We will explain this property later.

To find the representation of a given object in the morphing space we propose the following technique. First, we project the element on the axes of the space. This is done by minimizing the distance to the elements of one axis. All these projections result in a $n-1$-dimensional vector. We claim that this mapping is linear. In this case, the actual representation is a linear mapping of the found vector. Linear mappings can be represented by matrix multiplications. Thus, all we have to do is to find the representation matrix.

First we want to explain why minimizing the distance to the elements of one axis is linear: For linear morphing spaces there exists an object representation, such that the morphing function is a simple linear interpolation. This is due to the fact that the morphing space is a vector space, and in vector spaces all points between to given points can be obtained by linear interpolation.

Assume our distance function is a norm (which is no restriction, since all mathematical distance functions are norms). Thus, the distance function is itself linear, and the iso-distance surfaces are all similar. Minimizing the distance results in a projection with a fixed angle to the axis. These projections are linear.

As we have seen, the projection onto one axis is linear. And because the composition of linear operations is linear, the mapping from the element to the vector of projections is also linear.

Note, that we have assumed the distance function to be calculated on a representation of the elements, such that morphing between elements is represented by straight lines. This is the property we spoke about in the beginning of this section. The function d has to be a norm on the vector representation of the elements. It is not enough, that the morphing space is linear and the function d is a norm on another representation.

Assume the projection of an element A results in the vector $(a_1, \ldots, a_{n-1})$ while the actual representation of A is $(x_1, \ldots, x_{n-1})$. We are searching for the representation matrix M that connects those two representations:

$$M\mathbf{x} = \mathbf{a} \tag{5.32}$$

As we know from linear algebra, the rows of M consist of the images of the base vectors. Therefore, we obtain M by projecting the base elements $B_1, \ldots, B_{n-1}$ onto the axes.

Note that we cannot decide, whether the above linear system has a solution. This is due to the fact that the linear independence of $B_1, \ldots, B_{n-1}$ is not known, and not calculable so far. We suggest using the Singular Value Decomposition (SVD) to analyze M.

The SVD is a decomposition of a matrix into a product of an orthogonal matrix, a diagonal matrix and again an orthogonal matrix. This decomposition is always possible [Golub & Van Loan 1989]. The SVD sheds some light on the structure of a matrix. For most applications the values of the diagonal matrix (singular values) are of particular interest. If any of the singular values is zero, the determinant of the decomposed matrix is zero and the matrix not invertible. In our context, the above linear system would have no solution. If we replace any zero singular value by infinity, we can invert the diagonal matrix (the orthogonal matrices are invertible anyway). This way we find a best-approximation to the above linear system, no matter what the condition of M is [Press et al. 1992].

Furthermore we get a quality measure of our basis. The singular values represent the different scaling factors of the linear mapping. If these scaling factors differ substantially, the quality of the basis is obviously bad. If the scaling factors

are all the same, then the basis is orthogonal and to some degree optimal. As a quality measure we use the ratio between the maximum and minimum element of the singular values. A ratio of zero shows that the base elements are linear dependent.

5.5.4 Analysis in non-linear morphing spaces

In a non-linear morphing space we cannot use the above method, because the projection on the axes is not necessarily a linear mapping of the vector representation. However, we can treat the morphing space as any n-dimensional space and search for the minimum-distance element using a general minimization algorithm for n-dimensional spaces. The classical techniques assume that the minimum is unique, i.e. there is only one local minimum. In this case one can find the minimum by repeated minimizing along orthogonal lines in the space.

Powell's algorithm (see [Press et al. 1992] for instance) would be traditionally the method of choice for such cases. But note that morphing spaces differ somewhat from other vector spaces: In vector spaces one can specify a line by one base point and a direction vector. Conversely, in morphing spaces we need two base points (objects). If a sophisticated algorithm (such as Powell's) specifies some lines, along which one has to minimize, this causes the calculation of additional objects in the morphing space. But constructing objects is expensive and should be avoided as far as possible. Therefore we propose the following scheme to minimize in morphing spaces with a unique local minimum:

Given a morphing space Φ with the basis $B_0, \ldots, B_{n-1} \in \Omega$ constructed with a morphing function m and an element C, which has to be analyzed. Calculate $C_0 = m(B_0, B_1, w(B_0, B_1, C))$ and then interactively

$$C_i = m(C_{i-1}, B_{i+1 \bmod n}, w(C_i - 1, B_{i+1 \bmod n})) \tag{5.33}$$

With this, $C_{i \to \infty}$ converges against C.

The idea is to minimize along lines given by the actual best approximation and a base object. The index $i + 1 \bmod n$ results in cyclic usage of the base objects. This scheme converges due to the fact, that the respective direction vectors are linear independent. The advantage is of course that each step no object has to be predetermined to define the line along which one has to minimize.

Note that this trick works only when the distance-minimum is indeed unique in the morphing space. The above algorithm finds the first local minimum it "comes across". It will not find the global minimum if other local minima exist. For such cases more robust methods are needed.

We successfully adopted a simulated annealing scheme [Kirkpatrick et al. 1983] adapted to our needs: First, calculate $C_0 = m(B_0, B_1, w(B_0, B_1, C))$. In every step, choose $x \in [0, 1]$ randomly and calculate

$$D = m(C_{i-1}, B_{i+1 \bmod n}, x) \tag{5.34}$$

The probability to set $C_i = D$ is given by

$$p = e^{\frac{d(C,D) - d(C,C_{i-1}}{kT}} \tag{5.35}$$

Thus, if D is closer to the minimum than C_i according to the distance function d, we always use D as the actual approximation (in this case p is greater than one). If D is not closer than the result depends on a randomly chosen value in the interval [0,1].

Obviously, this scheme represents a simulated annealing technique in morphing spaces.

5.6 Spaces of meshes from morphing

The conceptual extension of the framework to more meshes is rather straightforward as compared to possibly non-linear morphing functions. Given meshes $\mathcal{M}_i = (V_i, K_i)$ a common connectivity K together with vertex sets $V(\mathbf{e_i})$ is established. The vertex sets form a base of a space, which is reflected by using canonical base vectors $\mathbf{e_i}$ as indices. A morphed shape $(V(\mathbf{s}), K)$ is represented by a vector $\mathbf{s} = (s_0, s_1, \ldots)$ reflecting the shares of the meshes $\mathcal{M}_0, M_1, \ldots$.

Not all techniques presented in this work are equally suited to be extended to more meshes. The correspondence problem discussed in Chapter 2 seems to be relatively easy to extend. All meshes are embedded in the given parameter domain, which leads to barycentric representation of the original vertices. If each set of original vertices V_i needs to be mapped to all other meshes $\mathcal{M}_j, i \neq j$ the complexity would grow quadratically with the number of meshes. However, this is not necessary if a remeshing strategy is used to generate a consistent mesh connectivity (see Section 3.5). This procedure generates the same set of vertices over all shapes, thus, the complexity is linear in the number of meshes times the number of vertices used in the remesh, which is the best we can expect. Concluding, the best way to generate the set $\{(V(\mathbf{e_i}), K\}$ is to embed all meshes in a common parameter domain (spherical or piecewise linear) and then remesh to the desired accuracy. This has been demonstrated by Michikawa et al. [2001] (see Figure 5.1).

The vertex path problem now extends to compute combinations of several vertex vectors. Linear vertex combination is easily extended:

$$V(\mathbf{s}) = \sum_i s_i V(\mathbf{e_i}) \tag{5.36}$$

Surprisingly, any technique involving rotations such as the ones explained in Sections 4.2 and 4.4 seem to be difficult to extend. Instead of interpolating the orientation one could compute the principal components (moments) of the shapes and align them with the canonical axes of the coordinate system. To extend the local morph approach explained in Section 4.5 the linear combination has to be applied to the Laplacian coordinates.

Applications of such spaces of meshes range from modeling and analysis of shapes to animation. Praun et al. have termed the synthesis-analysis part digital geometry processing (DGP) [Praun et al. 2001]. Modeling could be achieved be combining several shape (features) to yield the desired result. This has applications

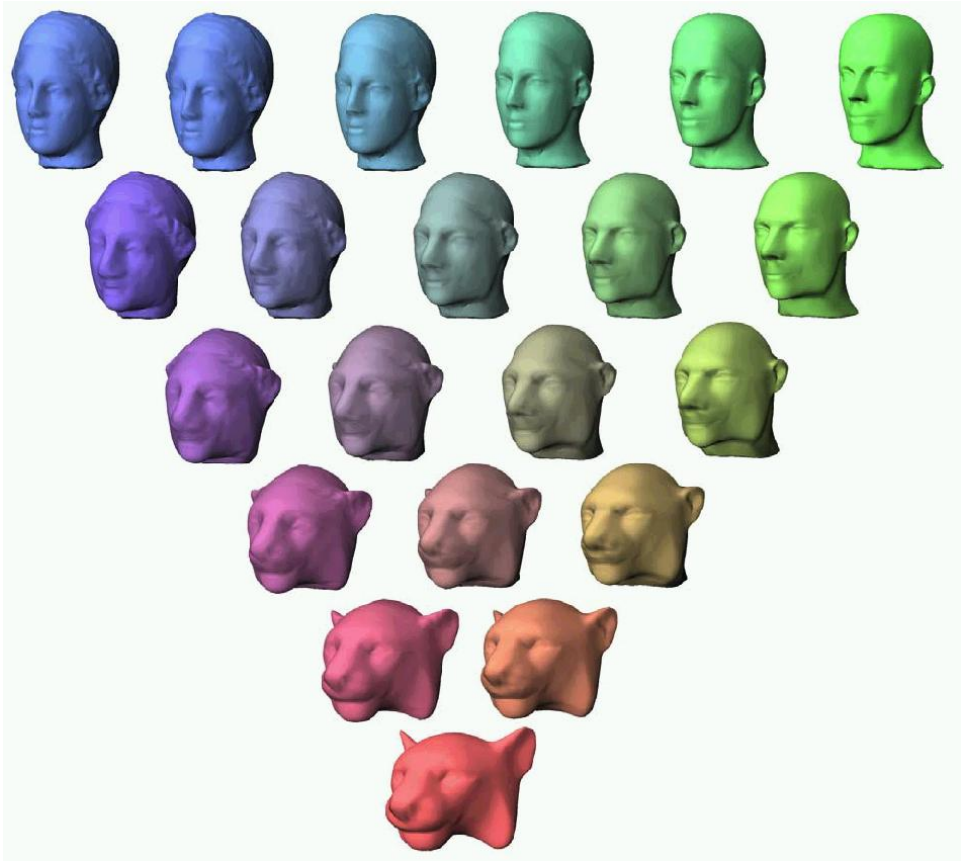


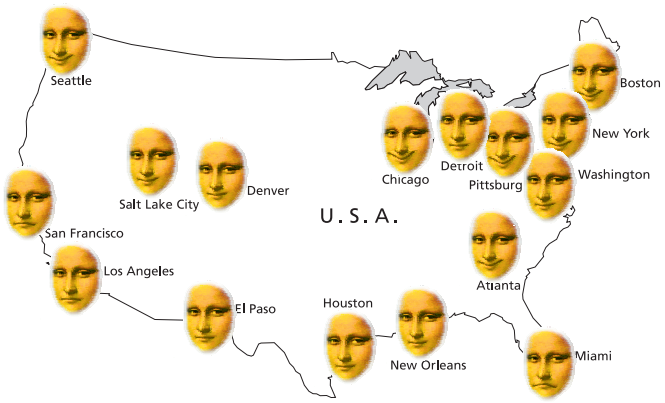
Figure 5.1: A space of shapes generate from three input shapes and linear interpolating their geometry vectors. Correspondence has been established using a coarse base domain and a multiresolution mesh. Reprinted from Michikawa et al. [2001].

in information visualization (see Chapter 6). Using techniques such as the principal component analysis, spectral properties of the mesh family can be explored.

The space of meshes $(V(\mathbf{e_i}), K)$ allows to represent animations as a curve $\mathbf{s}(t)$. This idea will be detailed in Chapter 7.

Chapter 6

Applications in Visualization



The visualization of scalar and multivariate quantitative data involves the mapping of data onto a visual scale. The principles of such a mapping of scalar data to scales of visual attributes are well-known for a number of basic scales. In color mapping, data values are mapped to appropriate hue of lightness values. Scatter plots are based on the principle of mapping data values to positions, or more exactly, to distances from an axis. Other variables often applied for such purposes are scale, form, and texture. Bertin [1983] describes a general methodology of how to select an appropriate mapping to these visual variables and how to combine them. Cleveland gives a ranking of their effectiveness [Cleveland 1985]. For multivariate data with two upto five dimensions more complex color and texture scales can provide solutions in some cases. For the visualization of local variations higher dimensions data glyphs have been proposed and been applied successfully. Chernoff faces [Chernoff 1973] and stick figures [Pickett & Grinstein 1988] are well-known examples for this visualization approach. Tufte gives a good overview of relevant visualization techniques and discusses their effectiveness in certain application areas [Tufte 1983]. Nevertheless, a generally accepted set of visualization rules does not exist.

All the techniques mentioned above represent fundamental approaches to the visualization of scalar and multivariate data. The general understanding is that no single of these visualizations is effective in all possible situations. One visualization may – and should of course – produce new insights and, by this, new questions which again produce the need of different views to the data. A good number of applications have proven that a user can gain knowledge about some unknown data more effectively when provided with highly interactive techniques [Rheingans 1992]. Techniques such as Focusing and Linking [Buja et al. 1991] extend this interactivity even further by connecting different views to the data using interactive feedback.

However, effective visualization still is very, very difficult. There are a number of reasons for this. First, a mapping of application data to these fundamental variables involves an abstraction. If the user is familiar with the idea of this mapping, he may understand the generated visualization good. However, often the application context is lost and the visualization which was applied to simplify the analysis of the data involves an analysis step or experience by its own.

Second, it is not easy to visualize a number of parameters using these fundamental mappings only. Usually, the visualization of multi-parameter data involves the generation of application specific models and solutions. General solutions and general scales do not exist for this purpose. Consequently, the user needs methods to define a visual scale for such applications very quickly and easily. Such methods do hardly exist to date.

Last, and may be even most important, it is still difficult to change visualization parameters and to produce a new, appropriate visualization intuitively. For example, to highlight a specific data value one has usually to modify the data filtering or to completely redefine the mapping of the data to visual attributes. A direct and interactive modification of local data mappings is hardly provided by any visualization technique today.

We have introduced a new approach for the visualization of scalar and multivariate data, which addresses the problems presented above [Alexa & Müller 1999b]. This approach is based on the direct and interactive specification of local data mappings.

6.1 Visualization by Examples

The general paradigm of our visualization approach is to enable the user to visualize some data by specifying the mapping of a small number of selected data values. We call this Visualization by Example.

This approach can be explained best with an example. A simple mapping of some scalar quantitative data to color can be defined by linking two arbitrary data values with appropriate color values. This results in a linear mapping. Further links can be supplied to adjust the mapping locally, resulting in a more sophisticated mapping function. The visualization of any local feature and its neighborhood is

directly and intuitively controlled by the user and may be easily changed when provided with appropriate visual attributes or objects on which the data may be mapped. Note that this strategy is fundamentally different from the selection of a color scale and applying this scale to all data.

While the direct and interactive linking of data values with visual representations is not new, it has not been combined with appropriate methods for the approximation of data mappings based on the supplied correspondences.

In the example presented above this approach seems simple and easy to follow. However, the generalization of this approach calls for a mathematical model describing the data objects, flexible visual scales, and the mapping between this values based on a small number of parameters and features.

In addition, the user will have some additional knowledge about the data in many cases, which can be exploited with our approach. Moreover, the user might want to highlight several data values by mapping them to special representations.

There exist approaches to describe data spaces based on mathematical models [Brodlie 1993]. However, in the context of this work mathematical models are proposed to describe the spaces of visual scales and morphing is used to construct the corresponding graphical objects for this purpose [Alexa & Müller 1998b; Müller & Alexa 1998]. This approach allows to define rich sets of useful visual representations from only a few graphical base objects. As such, this method provides the appropriate foundation for a Visualization by Example.

In the following section we will discuss the fundamentals and characteristics in more detail.

6.2 Visual representations from morphing

For our visualization technique, the visual representations have to be structured as a multidimensional space. That is, a visual object has to be element of an n-dimensional space and represented by a vector $r \in \mathbb{R}^n$.

For many visual scales such a representation is quite natural:

- Color can be represented by real values in $[0, 1]^3$, color scales might be represented by real numbers in $[0, 1]$.
- Size or position is naturally a real number.
- For textures one defines a number of real valued parameters, which control their appearance.
- In general: If the visual representations are used to depict quantitative data there has to be a reasonable understanding in terms of real valued vector spaces.

We have proposed a more flexible way to define spaces of visual representations [Alexa & Müller 1998b; Müller & Alexa 1998]:

Figure 6.1: A smiling scale produced by morphing a mona lisa face.

Figure 6.2: Another scale produced by morphing a mona lisa face.

Given a number of graphical objects of any class (images, polyhedra, etc.) we construct a space by morphing among these objects. Morphing between two objects produces one-dimensional visual scales. Two such scales are depicted in Figures 6.1 and 6.2. Here, the degree of smiling could be used to represent a scalar value. While this scale might not be as visually strong as e.g. the size of a dot it is easier to connect to the real-world phenomenon behind the data values. A smile is obviously representative for "good" values in the context of the data, while abstract representations need a legend, which has to be learned and remembered.

In this approach, a morph among multiple objects by performing several morphing operations between two objects subsequently defines a multidimensional space of objects (see previous chapter).

Since morphing is applicable to produce scales such as color, position, size, etc. we understand this to be a generalization of these techniques to define spaces of visual representations (e.g. glyphs or icons [Beddow 1990; Pickett & Grinstein 1988]). Note that our technique allows to represent glyphs or icons as an n-vector.

The strength of using morphing techniques to generate visual representations of data becomes evident when applied to multivariate data. As mentioned before, it has been proven difficult to find intuitive visual representations for multivariate data and multidimensional objects. By morphing among multiple objects one could visualize multivariate data as elements of a space of graphical objects.

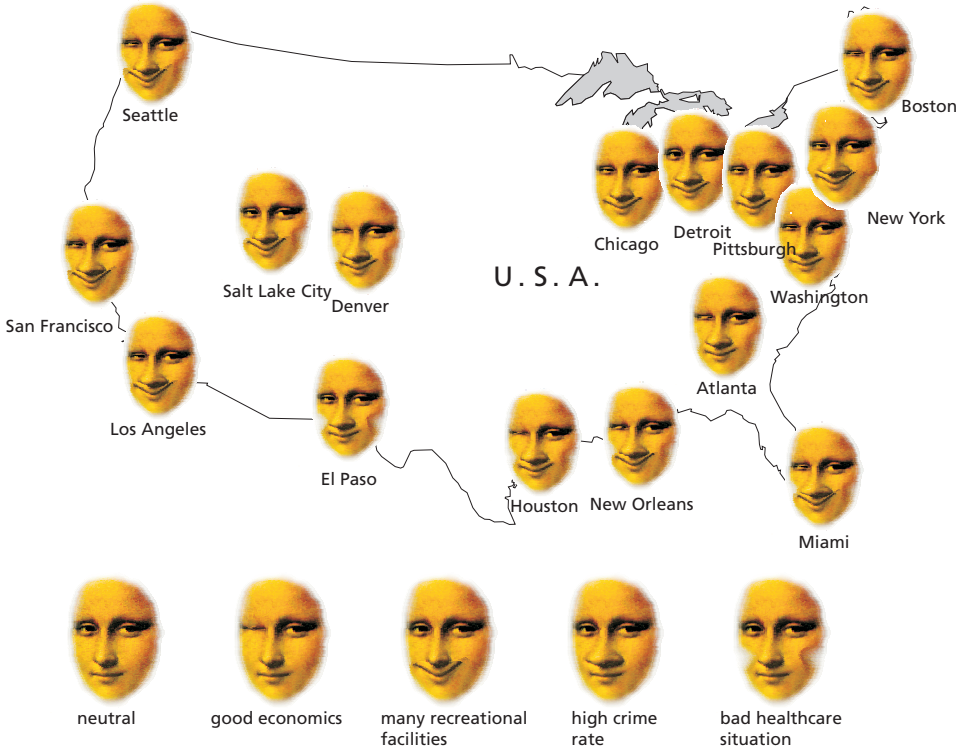


Figure 6.3: Mona Lisa faces as visual representatives of city rankings in the U.S.

6.2.1 Examples

Our visualization technique is based on the selection of a number of visual object attributes and the specification of visual prototypes representing a high data value against the global neutral object state. The visual prototypes are the base objects of the representation space. The correspondence between visual prototypes and data values induces a mapping from the data to coordinates, which are used to construct the actual visual representations.

We demonstrate the above procedure at application examples. In the examples we visualize data about U.S. cities from [Boyer & Savageau 1985] and the American Bureau of Concensus. Note that the examples are meant merely to demonstrate the technique of generating multivariate visual representations, and not to show the most effective way of communicating information.

In the first example we use warped images of Mona Lisa's face as base objects and visual prototypes (see figure 6.3). The idea of using faces as visual representations dates back to Chernoff [1973] and is advantageous due to human's native ability to recognize facial expressions. Note how morphing techniques add realism and additional degrees of freedom to this concept: An undistorted image is used as a representation of a neutral value. This image is distorted to represent good

economical situations (Mona blinks), many recreational facilities (Mona smiles), high crime rates (Mona's nose gets wider) and bad health care situations (Mona's cheeks tighten). Thus, the neutral face represents cities with bad economy, few recreational facilities, low crime rate, and good health care. In order to find data values that correspond to the intended meaning of the representations we simply scan the values for minima and maxima. The neutral face represents the smallest value in economics rating, recreation rating, and crime rating, but a high health care score. The other faces are based on the neutral face and add their specific characteristics (e.g.smile).

In the second example we use the 3D-model of a comic figure's hand (see figure 6.4). Note that navigating in the 3d-scene yields better access to the information than a single projection on 2D. The idea of using hands as glyphs is that, similar to faces, gestures are recognized intuitively by a human observer. However, we do not claim that this visualization is appropriate and effective for the given data. In this simple visualization example the neutral position is an open hand and each variate is represented by the flexion of one finger. A more sophisticated model could use hand gestures similar to those from sign languages. The process of connecting data values with objects is exactly the same as in the example above, as is the data set, now using the categories climate, economics, transportation and the arts.

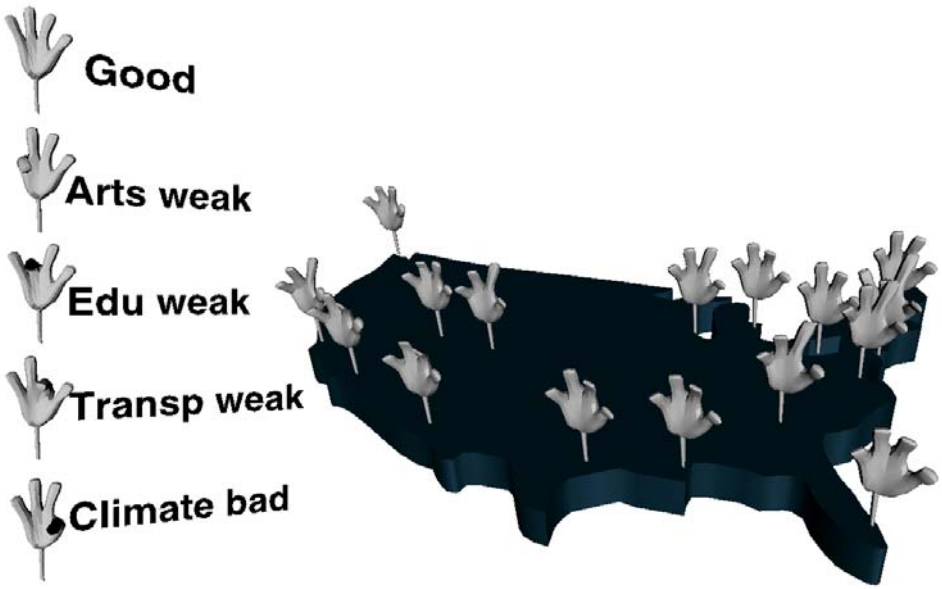


Figure 6.4: 3D hand glyphs generated by shape interpolation for the visualization of city rankings.

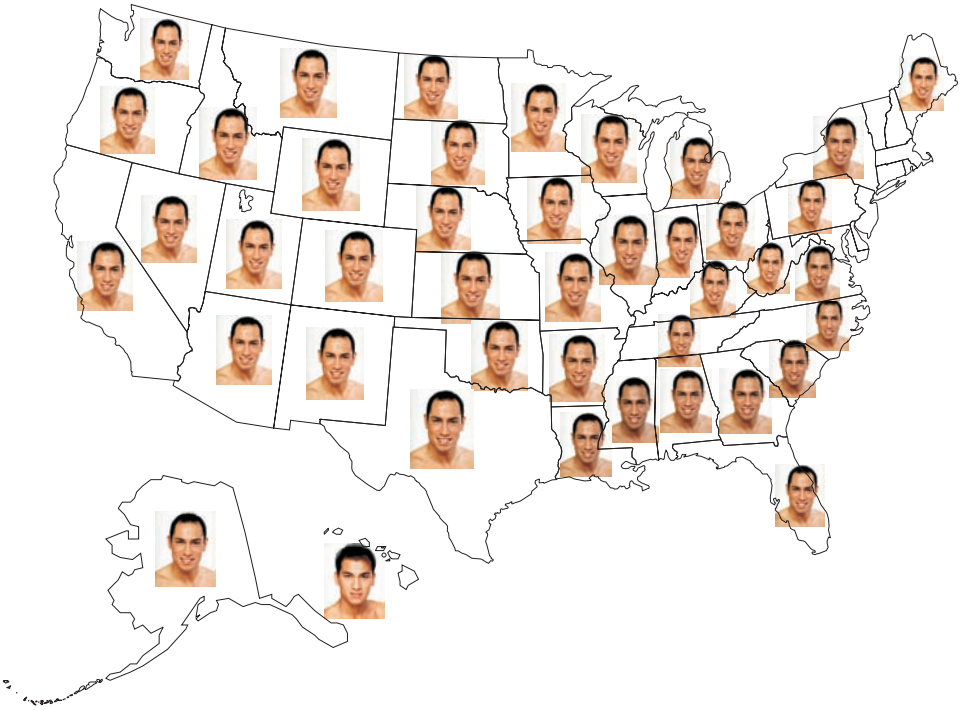


Figure 6.5: Ethnic distribution in the USA.

The third example shows the distribution of ethnic in the USA (see Figure 6.5). For each type of ethnical origin one prototype faces was used. Mergers of these faces represent the relative amount of people living in a particular state. This example shows how meaning of data could be connected directly to the visualization primitive being used.

6.3 Mapping Data to Coordinates

The main idea of our approach is to let the user define several relations between data values and graphical representations. These correspondences are used to construct a mapping from data to visual representations. We want to allow the user to define any number of correspondences, usually beginning with only a small number of correspondences. Depending on the application the user might decide to generate an affine mapping in any case, or, if no simple affine mapping can be found, to accept a non-linear function to depict the mapping.

To define a single correspondence, the user first chooses a data value and then searches the space of graphical representations for a suitable element. That is, the user gives an example for the intended relation between data and visualization.

We denote the space of data values as V^d, i.e. each vector consists of d variates. Assume a number e of data vectors $\mathbf{q_0}, \mathbf{q_1}, \ldots, \mathbf{q_{e-1}}$ should be mapped to a coordinate $\mathbf{r_0}, \mathbf{r_1}, \ldots, \mathbf{r_{e-1}}$ in the space of visual representations. If $e \leq d$ then an affine mapping $a : V^d \rightarrow \mathbb{R}^n$ exists that satisfies $a(\mathbf{q_i}) = \mathbf{r_i}$ for all $i \in 0, \ldots, e-1$. However, if $e > d$ that mapping does not necessarily exist.

We suggest to use a linear mapping whenever possible, i.e. in case $e \leq d$. The reason for this is that visual scales are still meaningful, properties of the data variates are preserved in the visualization, and the order of the data values is not changed.

If e is greater than d we offer three choices:

1. An affine mapping that fits the given correspondences as far as possible.
2. A non-linear mapping that satisfies all relations by locally changing a linear approximation.
3. A non-linear mapping that satisfies all relations by globally interpolating the relations.

The second and third mapping are constructed using the same approach. Both need an approximation of the affine mapping similar to the one used in the first approach. In the upcoming subsection we explain how to calculate that affine mapping, independent of the number of given relations.

6.3.1 Finding an affine mapping

We want the mapping a to be represented by a matrix multiplication. Thus, we are searching for a $n \times d$ matrix A that maps from V^d to coordinates in the space of visual representations. In case $e \leq d$, A has to satisfy the simultaneous equations

$$\begin{array}{rcl} A\mathbf{q_0} & = & \mathbf{r_0} \\ A\mathbf{q_1} & = & \mathbf{r_1} \\ \vdots & = & \vdots \\ A\mathbf{q_{e-1}} & = & \mathbf{r_{e-1}}, \end{array} \tag{6.1}$$

in case $e > d$ we like to minimize the residual

$$(\|A\mathbf{q_0} - \mathbf{r_0}\|, \|A\mathbf{q_1} - \mathbf{r_1}\|, \ldots, \|A\mathbf{q_{e-1}} - \mathbf{r_{e-1}}\|) \ .$$

We now first solve the first case. The techniques we employ here will automatically produce a solution to the second case.

If we look at the i-th row $\mathbf{a_i}$ of A we get the simultaneous equations

$$\begin{array}{rcl} \mathbf{a_i q_0} & = & r_{0_i} \\ \mathbf{a_i q_1} & = & r_{1_i} \\ \vdots & = & \vdots \\ \mathbf{a_i q_{e-1}} & = & r_{e-1_i}. \end{array} \tag{6.2}$$

We define the $d \times e$ matrix

$$B = \begin{pmatrix} - & \mathbf{q_0} & - \\ - & \mathbf{q_1} & - \\ & \vdots & \\ - & \mathbf{q_{e-1}} & - \end{pmatrix} \tag{6.3}$$

to rewrite the simultaneous equations in 6.2 as a matrix equation:

$$Ba_i^T = (r_{0_i}, r_{1_i}, \cdots, r_{e-1_i,})^T \tag{6.4}$$

The solutions of these n systems of linear equations yield the rows of A. In order to solve one of these systems we use the Singular Value Decomposition (SVD, [Golub & Van Loan 1989]). The SVD has several nice properties that are interesting for our problem [Press et al. 1992]

1. It gives a stable solution in the quadratic case, even in the presence of degeneracies in the matrix.

2. It solves the under-specified case in a reasonable way, i.e. out of the space of solutions it returns the one closest to the origin.

3. It solves the over-specified case by minimizing the quadratic error measure of the residual.

Using the SVD we can compute all rows of A and thus have found the affine mapping we were searching for.

6.3.2 Non-linear mappings

We want to find a mapping a that satisfies all equations $a(\mathbf{q_i}) = \mathbf{r_i}$. This could be seen as a scattered data interpolation problem where we try to find a smooth interpolation between the values $\mathbf{r_i}$ given at locations $\mathbf{q_i}$. Contrary to some other application domains of scattered data interpolation, we deal with different and high

dimensions of the vectors and typically the number of relations is close to the dimension of the input data.

As explained before, it seems desirable to have an affine mapping from the data values to the space of visual representations. Therefore, we always start with a linear approximation of the mapping (as calculated in the previous section) and then fit the relations in the mapping by tiny adjustments. For these adjustments we use radial sums. The idea of combining an affine mapping with radial sums for scattered data interpolation is considered in e.g. [Arad & Reisfeld 1994] and [Ruprecht & Muller 1995] (for two-dimensional vectors, only).

Hence, we define a by

$$a(\mathbf{q}) = A\mathbf{r} + \sum_j \mathbf{w_j} f(|\mathbf{q} - \mathbf{q_j}|), \mathbf{q} \in V^d, \mathbf{r} \in \mathbb{R}^n \tag{6.5}$$

where $\mathbf{w_i} \in \mathbb{R}^n$ are vector weights for a radial function $f : \mathbb{R} \to \mathbb{R}$. We consider only two choices for f:

1. The Gaussian $f(x) = e^{-x^2/c^2}$, which is intended for locally fitting the map to the given relations.

2. The shifted log $f(x) = \log\sqrt{(x^2 + c^2)}$, which is a solution to the spline energy minimization problem and, as such, results in more global solutions.

We compute A beforehand as explained in the previous section. Thus, the only unknown in (5) is a pure radial sum, which is solved by constituting the known relations

$$\mathbf{r_i} - A\mathbf{r_i} = \sum_j \mathbf{w_j} f(|\mathbf{q_i} - \mathbf{q_j}|) \tag{6.6}$$

This can be written in matrix form by defining

$$F = \begin{pmatrix} f(0) & f(|\mathbf{q_0} - \mathbf{q_1}|) & \dots & f(|\mathbf{q_0} - \mathbf{q_{e-1}}|) \\ f(|\mathbf{q_1} - \mathbf{q_0}|) & f(0) & \dots & f(|\mathbf{q_1} - \mathbf{q_{e-1}}|) \\ \vdots & \vdots & \ddots & \vdots \\ f(|\mathbf{q_{e-1}} - \mathbf{q_0}|) & f(|\mathbf{q_{e-1}} - \mathbf{q_1}|) & \dots & f(0) \end{pmatrix}$$

and separating 6.6 according to the n dimensions of $\mathbf{r_i}$ and $\mathbf{w_i}$:

$$F \begin{pmatrix} w_{0_i} \\ w_{1_i} \\ \vdots \\ w_{e-1_i} \end{pmatrix} = \begin{pmatrix} r_{0_i} - \mathbf{a_i r_0} \\ r_{1_i} - \mathbf{a_i r_1} \\ \vdots \\ r_{e-1_i} - \mathbf{a_i r_{e-1}} \end{pmatrix}, i \in 0, \dots, n-1 \tag{6.7}$$

Again, we solve these n equations by calculating the SVD of F. This time we are sure that an exact solution exist, because the solvability for the above radial functions f can be proven [Dyn 1989].

6.4 Results

We will demonstrate the techniques at two examples. These examples show two principally different application scenarios:

- The first example shows the mapping from multivariate data onto low-dimensional visual representation. That is, the dimension of the data is much higher than the dimension of the representations.
- The second example shows a mapping from scalar data onto either basic or more complex, multi-parameter representations. Here, specific aspects of the scalar data set are mapped to a specific channel of the visual attribute enhancing the expressiveness of the visualization.

6.4.1 Visualizing city rankings

In this example we visualize an overall (scalar) ranking of cities in the USA. Suppose we want a visual aid for a decision which of the major cities would be nice to live in. In order to quantify the different amenities and drawbacks of these cities we use data from "The places rated almanac" [Boyer & Savageau 1985]. This data contains values for nine different categories. That means, we need to project nine-valued vectors onto scalar values.

To visualize the ranking of the cities we use a Chernoff-like approach. The faces are generated by morphing among a standard set of facial expressions (as explained in the following chapter). In this example we make use of only a smile and a grumble, defining a one-dimensional visual scale. Thus, the degree of smiling represents the living quality determined by a combination of the nine data attributes from [Boyer & Savageau 1985].

One way to find this mapping might be to inspect the nine different categories and try to find some weights for the values. This requires not only to define nine values, also the correlation to the outcome of this mapping does not take into account the user's knowledge about the cities.

A more intuitive approach is to allow the user to supply a ranking based on personal experience. Remark that a ranking of a subset of all cities is sufficient. In figure 6.6 only three examples were given to generate an affine mapping. Namely, Chicago was thought to be nice to live in and was mapped to a smiling face, whereas Miami was unacceptable and mapped to a grumble. Additionally, Washington appeared nice but expensive and, therefore, mapped to a neutral face.

6.4.2 CT scan data

In this example, we inspect CT scan data from the "Visible Human"-project. The data is given as 16-bit data values on a 512 by 512 grid. A standard linear mapping of the relevant CT data to gray values is depicted in figure 6.7. Note, that this image could be produced by picking the two boundary values to define an affine map.

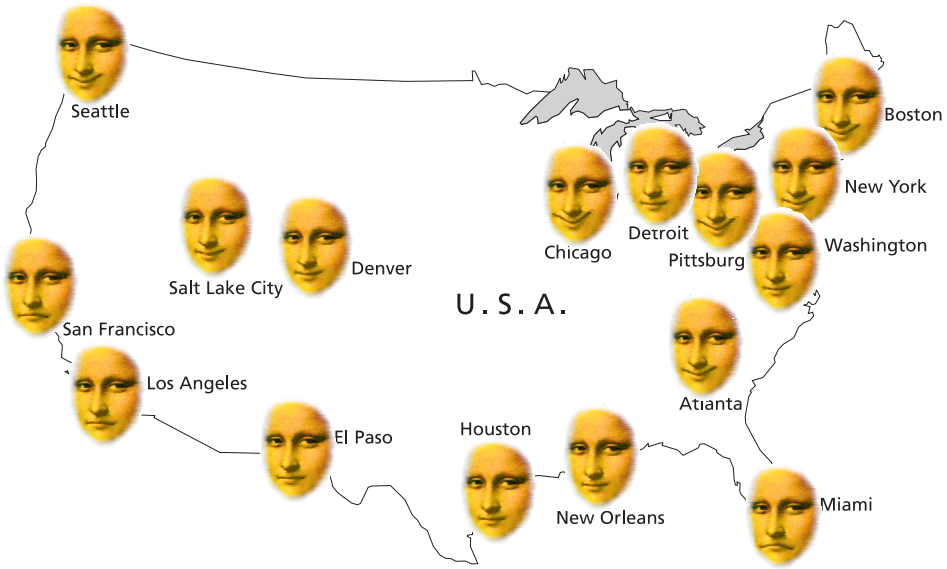


Figure 6.6: Cities of the united states represented by mona lisa faces. The representation is generated from 9-valued ranking vectors. The mapping was defined by mapping Chicago to a smile, Washington to neutral face, and Miami to a grumble.

In figure 6.7 the soft tissue is display relatively bright. We can adjust this for a better distinction of bones and soft tissue by simply selecting one of the data values from the soft tissue and assigning a dark gray to it. This time an affine mapping is a bad choice, because the three correspondences cannot be satisfied. Instead, we fit the mapping globally to the data value - gray value pairs by using radial basis sums with the shifted logarithm as the radial function. The resulted is depicted in figure 6.8 and clearly shows the advantage in comparison with figure 6.7.

If we take a closer look at figure 6.8 we find a brighter substructure in the stomach. We would like to bring this region of data values to better attention in the visual representation. We do this by mapping a data value of this region to a red color. That is, instead of using gray values in the visualization we now use RGB color. Note, that it is not necessary to use specific two-dimensional color scales: We simply specify which data value maps to which RGB triple. The gray value representations of the three correspondences defined earlier are mapped onto corresponding RGB values. The resulting mapping is shown in figure 6.9. Note, how the empty structures are colored in the complementary color of red. This gives a nice distinction of empty spaces and tissues.

6.5 Conclusion

We present a new approach to the construction of visual scales for the visualization of scalar and multivariate data. Based on the specification of only a few correspondences between data values and visual representations, complex visualization mappings are produced, hereby introducing a Visualization by Examples.

This approach exploits the user's knowledge about the data in a more intuitive way. Moreover, the user is enabled to adapt the visualization interactively and easily. The technique of Visualization by Examples can be used in combination with any visual representation

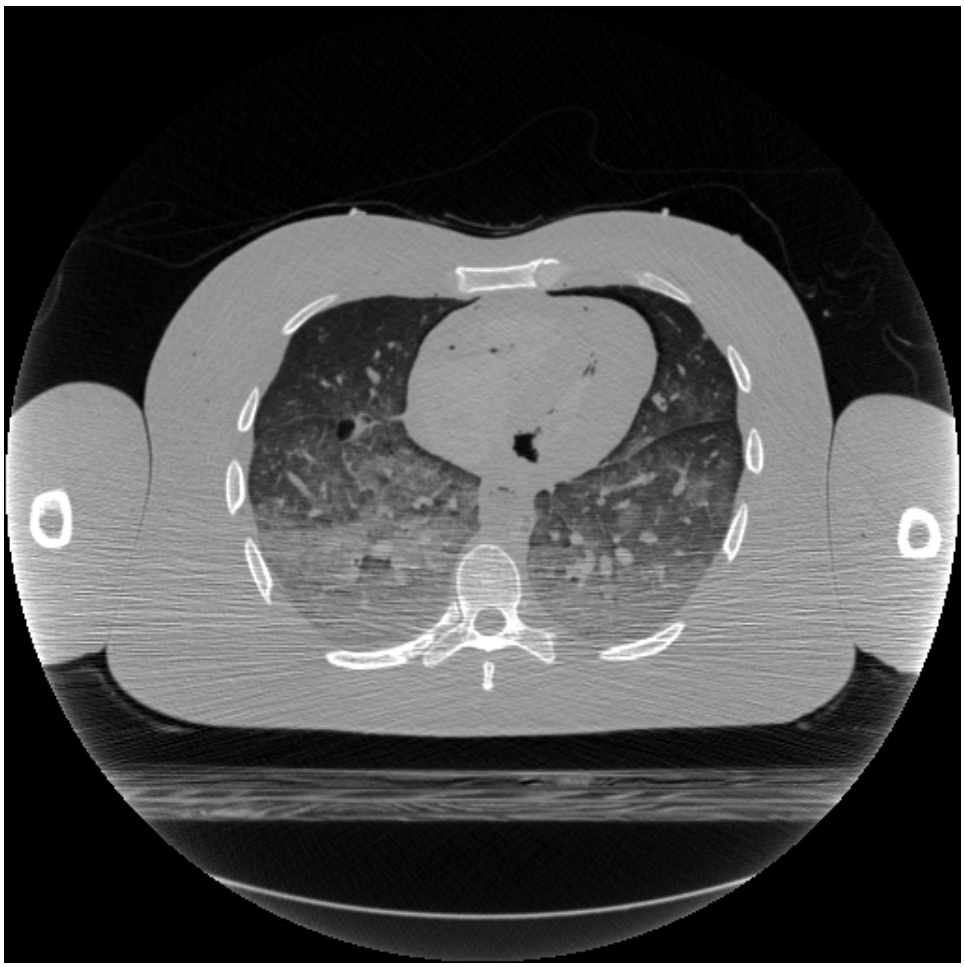


Figure 6.7: The CT-scan of the chest of a man. This image is generated from the raw CT-data by linearly mapping the range of useful CT-data values to a grey-scale

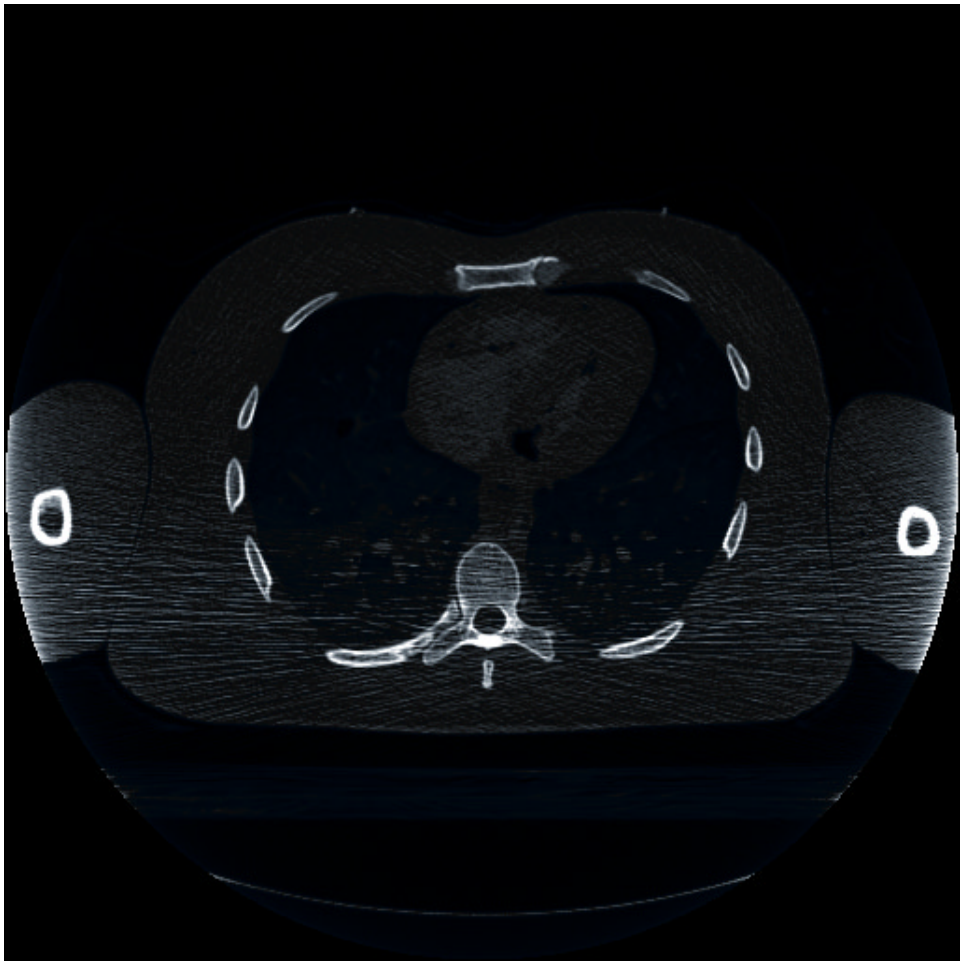


Figure 6.8: Here, the CT scan was generated by a mapping defined from three correspondences. The background was mapped to black, the bones were mapped to white, and the soft tissues surrounding the lung were mapped to dark grey.

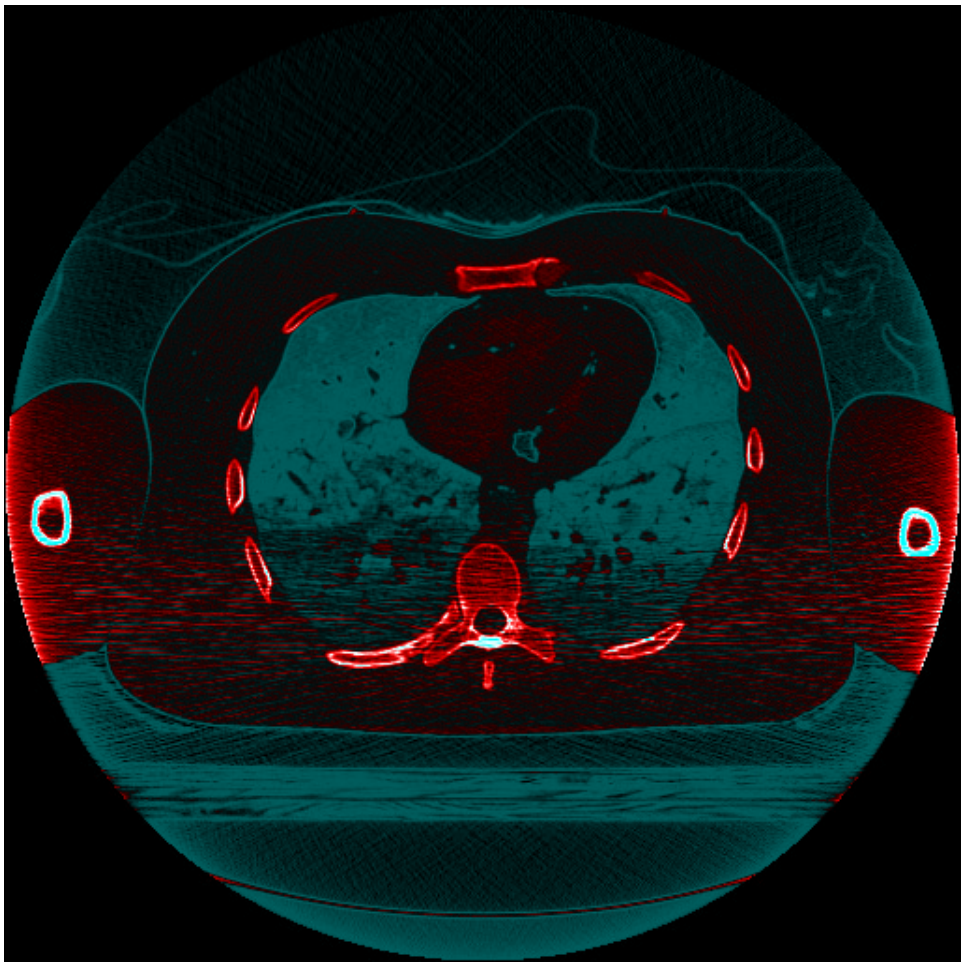


Figure 6.9: This image demonstrates the benefits of displaying scalar data with multidimensional visual representations. In addition to an already defined gray-scale, the soft structures of the bones were mapped to red color.

Chapter 7

Applications in animation

Computer animation has evolved to a standard technique in Computer Graphics. In the last decades, a number of different animation techniques have been developed. Starting from the standard frame-by-frame animation introduced by Disney and others, key-frame animation has established itself as the standard technique for describing time-dependent scenes in Computer Animation. Instead of describing every single frame only a sequence of principal frames - so called key-frames - is defined and additional frames are generated by interpolating between two consecutive key- frames using in-betweening. Elaborated techniques have been developed to allow for the automated generation of physically-based behavior, namely kinematics and inverse kinematics, and today animation systems are more and more coupled with simulation engines to facilitate the production of complex and realistic animations. However, key-frames and the concept of describing object states at specific times remain the fundamental philosophy for representing animations and time-dependent aspects in virtual scenes.

Despite its widespread use, the concept of merging object and geometry descriptions has a fundamental problem. While it is easy to specify, design, or output an animation in terms of key-frames, it is difficult to manage or change the time-behavior since all meta-information is lost. In particular, the following problems connected to geometric key-frame animations can be stated:

- Redundancy: A complete object description has to be recorded for each key-frame, even if parts of the object do not change at all. Additionally, repetitive patterns result in repeating the geometry description. Consequently, animation sequences are usually very large and hard to apply in streaming appli-

cations. The compression of such sequences is a problem which is under intense investigation of the Computer Graphics research community today.

- Modification: Exchanging an animated object in a scene while reusing the once specified animation at the same time is an involved task, even though most of the necessary information should be available. In general, after introducing the new model, it has to be animated again by hand. Similarly, it is almost impossible to make use of a once defined animation and to extend or exchange the object's behavior. The aspect of reuse has been addressed for specific object domains. One approach is standardized parameterization of an object and its possible behaviors, thus allowing for the exchange of both, geometry and animation (e.g. Humanoid Animation [Web3D Consortium 1999a] or Facial Animation in MPEG-4 [Ostermann 1998]). Similar but more general is the idea of animation elements [Dörner et al. 1997], general object hierarchies with a defined interface. However, both approaches do not provide a general solution to this problem.

- Level of detail: Another problem of high impact is the reduction of scene complexity in interactive applications. LOD concepts such as progressive meshes [Hoppe 1996] allow static objects to be fitted to the display requirements. Recent techniques try to provide a view-dependent level of detail on static object based on mesh simplification techniques or sometimes by exploiting progressive transmission and decompression schemes. However, if the objects are animated these standard techniques may fail or not be applicable at all. Standard LOD hierarchies could not be applied for animation since this would result in even a much higher redundancy since specific LOD hierarchies would have to be provided for all key-frames. Key-frame interpolation also would become more difficult since an interpolation between objects of different LOD would have to be supported.

  Still, geometric simplification exploits just spatial coherence, while animations exhibit additional temporal coherence. Surprisingly, the application of a LOD concept for animated geometry and animation itself has not been discussed in the literature to the best of our knowledge. For the same reasons small, static geometry features are omitted in standard LOD techniques, small temporal features should be a target of LOD approaches also. Here, an interesting potential for reducing scene complexity is hidden.

Key-frame animations and related concepts inhibit a compound of geometry and animation in their scene description. For all the problems stated above, this composition represents the main obstacle. It makes an easy exchange of either geometry or animation in a scene description impossible and makes the application of LOD concepts and abstraction difficult.

Here, we address this problem and present an alternative representation of animation sequences based on principal animation components, thus decoupling the animation from the underlying geometry [Alexa & Müller 2000]. The idea is find

or use a set of basis shapes and represent each (key) frame as linear combination of the basis objects. In the spirit of this work the basis objects form a space and the animation is a curve through this space.

We present two ways of representing animations in this way. First, we start from a suitable basis of shapes. The basis shapes and their blends are used to author the animation and the animation is then naturally represented in terms of the basis. Second, starting from a key frame animation we try to find a suitable basis consisting of only a few shapes.

7.1 Building animations using base shapes

We demonstrate the idea of building animations from an existing set of base shapes at the example of facial animations [Müller et al. 2000; Alexa et al. 2001].

Models for Facial Animation have been of much interest in Computer Graphics in the last years. One of the first examples was presented by Parke [1979], who used a selected number of key expression poses. Specific expression poses could be generated by interpolating between these key expression poses. Interpolating between these expression poses would then create facial animations. However, this approach has been criticized to allow only a generation of a limited range of facial expressions and to request a significant amount of modeling time to specify the key expression poses [Parke 1982]. The need for more realism in facial animation lead to the development of Facial Animation techniques based on physiological models. Waters [1987] introduced a model for facial animation that incorporates skin and virtual muscles corresponding to the ones in a human face. Activation of these muscles results in different facial expressions. For the control of these muscles FACS, a classification scheme for facial expressions [Ekman & Friesen 1978], was applied. Magnenat-Thalmann et. al. developed a pseudomuscle-based model in which the parameters control abstract muscle actions (AMA, [Magnenat-Thalmann et al. 1988]). While the approach is similar to the one of Water, FACS actions are exchanged here for more complex actions. While the application of muscle-tissue based facial expressions may result in very realistic images, animations are still not easy to control. Moreover, the corresponding models are much to complex and not flexible enough for an application in the context of real-time animation.

Facial animation techniques have been suggested for bandwidth compression in tele-communications for video images of a human speaker. Here the idea is to transfer a geometric representation of the speaker's head and face and their movements rather than a complete image of the speaker every second [Parke 1982]. However, until now prototype systems in this area succeed in making use of the head and shoulder's geometry. The description of speakers in terms of geometric models and facial expressions has gained much interest in the context of MPEG-4 [ISO JTC/WG11 1997]. MPEG-4 allows the definition of data streams with 2d and 3d objects. Moreover, MPEG-4 specifies a set of face animation parameters [Ostermann 1998]. Each one of these parameters corresponds to a particular fa-

cial action deforming a face model in its neutral state. The underlying approach is again based on a physiological model. While the integration of facial animation control techniques in this streaming standard gives a much higher degree of flexibility, solutions for the modeling of animations and specific expressions are not presented.

7.1.1 Representing facial animations

The main use of facial animations in our context is communication. Basically, we are not interested in being able to represent every possible state of a human face. But, we might want to represent facial expressions that are not realistic in order to generate cartoon-like characters. For these two reasons, most of the documented parameter sets used to describe facial expressions are not adequate: First, they usually exhibit a very high degree of granularity and detail, more than might be necessary for the sole purpose of communication. Second, they are derived from the human face and have problems to represent unrealistic facial expressions.

When designing conversational user interfaces, the optimal origin for creating facial expressions is a neutral, emotionless representation of a face, to which certain features can be added, such as:

- Movements of the mouth representing phonemes (so called visemes)
- mimic (for example blinking, raising eyebrow, smiling, grumbling)
- overlaid moods (i.e. friendly, sad)

The systems should add these features non-exclusively, i.e. phonemes, mimics and moods can be combined and result in an addition of features. Also, the number of different features should be as small as possible while still satisfying the needs of human communication. Obviously, the quantity of each feature has to be controllable. We will describe the single features we have used for our prototype system in detail.

We will use the quantities of the above mentioned features (visemes, mimic, moods) as the representation for facial expressions. In the following we will explain our idea of connecting this representation to geometries.

Our approach somewhat resembles the idea of Parke [1979]. We use not only one geometry for the neutral state of the face, but an additional geometry for each feature. But, instead of just blending between these geometries as in [Parke 1979] we define a vector space of geometries. This vector space has the neutral face as zero element and the differences of the features' geometries to the neutral face as base vectors. This alleviates one key problem of the blending technique: Features may be added independently to the neutral face. For example, the designer can combine speaking an "A" with a smile, where the blending technique would allow only to trade off between speaking and smiling.

Also, the generation of the geometries can be eased with modern techniques. We can produce several laser scans of a human model and use the morphing techniques composed from the methods discussed in Chapters 2, 3, and 4. to produce a consistent geometry. But even if the set has to be modeled by hand, several advantages warrant the task.

Note that the movements of vertices for each feature are linear, which is obviously not correct for every facial expression. However, in our experiments this simplification has proven to be sufficient. It seems that linear approximations are not too bad, because of the small distances and speed with which vertices move. If necessary one could model movements of vertices in a more elaborated way and use eigenvector analysis to decompose the movement into linear parts.

Distinct facial features

In our system we like to allow a designer to start forming a neutral face and to add several expressions. The main goal of our work is to communicate in an easy readable and easy understandable way. Facial expressions include a huge potential in terms of conciseness and a wide spectrum in terms of communication statements. The given principles of different communication channels such as speech and mimics contribute to an easy understandable communication through redundancy (e.g. synchronous appearance of speech and mimics) and thus prevent misinterpretation. We need a collection of features which allows to conveniently and intuitively design communication by lifelike facial animations. We do not strive for completeness in the sense of being able to represent the whole range of possible facial expressions. But, the variety of facial expressions resulting from the defined features must be diverse enough to be accepted by a human observer.

Obviously, such a set has to be found by analyzing special communication requirements, such as:

- the syntactical, the semantical as well as the pragmatical context of facial expressions (e.g. different meanings and interpretations of facial expressions according to several cultural agreements)
- special communication requests and special facial expression codes of the intended audience (i.e. target group).

After this analysis step, a base of elementary prototypes of facial expressions has to be defined. This base can be refined in further steps by repeated experimentation.

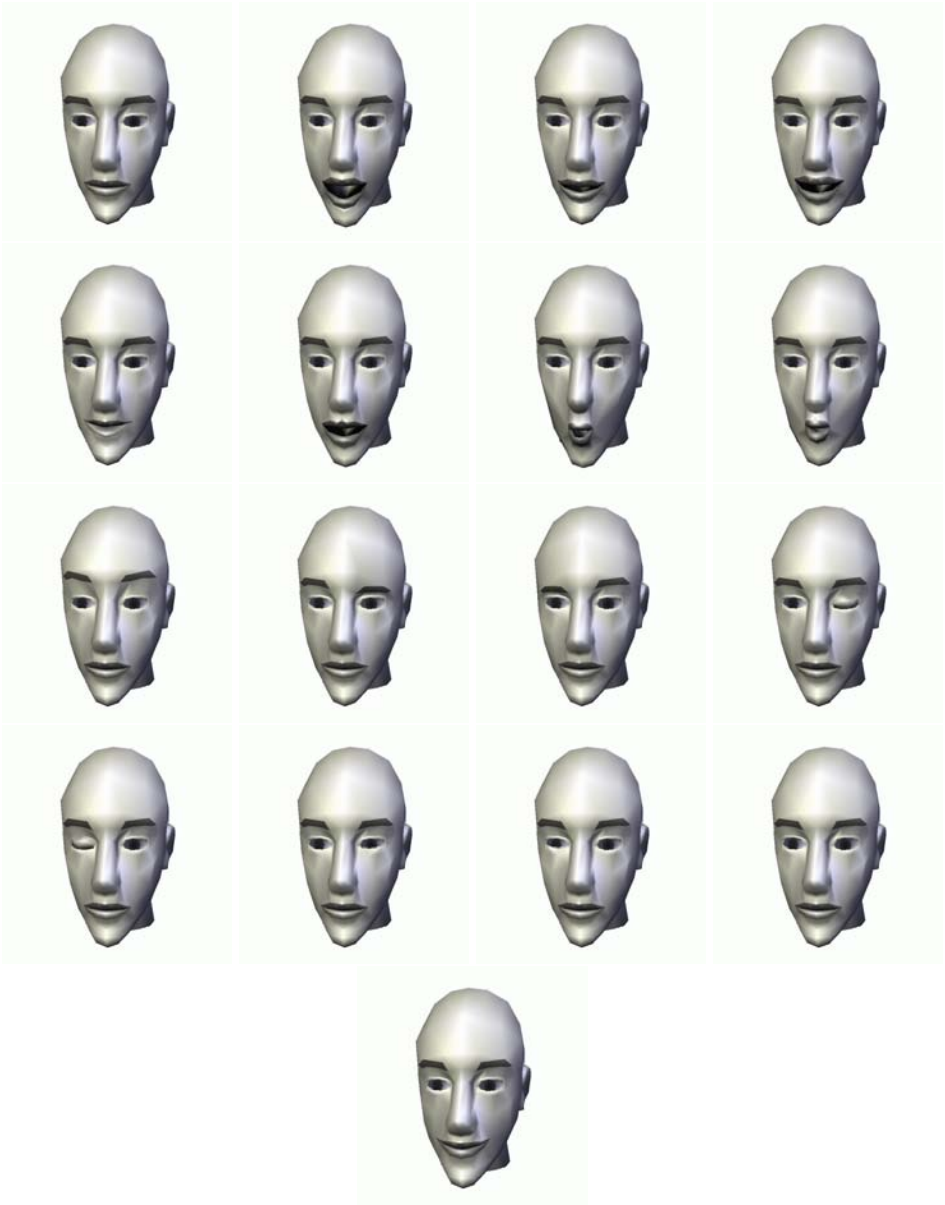


Figure 7.1: Some basic facial expressions

Compact representation of the Animation

A complete representation of an animation consists of the geometry and the change of this geometry over time. In our scheme such a complete representation is given by

- the vertex edge-topology of the geometry
- vertex attributes of the neutral face.
- vertex attributes of the feature facial expressions or the differences to the neutral face
- key-frames given by a feature vector together with a time stamp.

Representing vertex-edge topology in a compact way is a current research topic. We are aware of schemes that guarantee less than 2 bit per vertex, also enabling a compact way of representing vertex positions. In general, coding the vertex attributes is linear in the number of vertices. For representing the features we can exploit the fact that they differ only a small amount from the neutral face. Thus, all values are small and we can use a small number of quantization levels resulting in compact representations. Anyway, the cost for coding the geometries is independent from the actual animation and might be used for a number of different animations.

The key-frames consist of a time stamp (real valued number) and the representation vector (n real valued numbers). Practice shows that a single precision floating point number (2 byte) is sufficient for the time stamp. For the components of the representation vector not more than 6-8 bits are needed. This sums up to less than 20 byte per key-frame. Even if one likes to specify 25 key-frames per second, the stream is represented with less than 4kBaud, a transfer rate each modem can easily provide. Note that a rate of 25 frames per second eliminates all visual problems originating from interpolation issues.

An important point is that an animation's representation is independent from geometry but based on quantities of facial expressions. Of course, the animation makes sense only with a specified geometry, but, this geometry is exchangeable. In the following two sections we will show how to exploit this idea in two ways. We will show the effects of

- changing the representation of an animation (independent of the geometries used)
- using different geometries for the facial expressions (independent of the actual animation sequence).

7.1.2 Altering and combining animations

In this section we show why the representation of facial expressions in terms of quantities of different facial expressions is useful. For this section it is convenient

to represent the animation as a vector over time rather than defining key frames. Simply speaking, the animation is defined by the quantities of saying-an-a over time, smiling over time, and so on. Formally, an animation sequence is represented by $\mathbf{r}(t) = (r_1(t), \ldots, r_{n-1}(t))$. Note that this is only a conceptual difference and we get back to a key-frame animation by simply discretizing $\mathbf{r}(t)$. In the following, we like to exploit the idea of filtering $\mathbf{r}(t)$ and combining different sequences $\mathbf{r}(t)$ and $\mathbf{s}(t)$.

Filtering animations

Given an animation sequence $\mathbf{r}(t)$. Each of the components describes the influence of one of the facial base expression over time. First, lets look at the effect of operations on only one component at the example of smiling: By adding a constant to the component representing smiling or multiplying it with a factor greater one we make the face 'happier'. Conversely, by subtracting a constant or multiplying with a factor less than one the face appears less 'happy'.

For most geometries we should make sure that each the scalar $r_i(n)$ is in $[0, 1]$. Another way of altering the animation would be limiting the range of values for a component. In this way, one could e.g. set a minimum smile, so that the face is always smiling.

The next level of operations would be to intertwine different components. That is, we could e.g. exchange grumbles by smiles and vice versa. If we only look at linear operations on the components, the idea of changing each component based on the quantity of all components could be modeled by a matrix multiplication.

Even more flexibility is achieved by also involving time. If we let these changes be triggered by some events on the components really interesting effects are possible. For example, we could add a smile following each blinking.

Combining animations

Combining animations is the key to powerful and convenient authoring. We will start to explain the idea by looking at the combination of two animations, $\mathbf{r}(t)$ and $\mathbf{s}(t)$. Basically, we understand combination as a weighted sum of respective components over time: $\lambda_r \mathbf{r}(t) + \lambda_s \mathbf{s}(t)$. As in the previous section, clipping the vector components to $[0, 1]$ might be necessary. Additionally, a phase shift τ might be useful $\lambda_r \mathbf{r}(t) + \lambda_s \mathbf{s}(t + \tau)$, which also introduces the idea of smooth blending from one animation into another (fading one animation out, the other one in).

Some useful example for the case of combining just two animations is this: Typically, a human is blinking from time to time to moisten the eye. One could create an animation, where the face is just blinking from time to time. This could be added to any other animation, thus, eliminating the need to author blinking in every animation.

Of course, the idea of combining animations fosters the paradigm of a component based authoring system. Animators would author only small components and,

once a base of useful components is established, just plug them together as needed. This would be best done with a hierarchical approach. Ideally, a text to speech system would serve for constructing the phoneme/viseme part of the animation.

The idea of animation components is also important for the design of interactive systems. According to user input, the system would pick pre-authored components to react flexibly. Smooth transitions from one animation to another also help to change behavior quickly, but still visually pleasant and realistically.

7.1.3 Streaming and displaying animations

In this section we will show the effects of changing the geometry used to display. Using different geometries serves several needs: First, we can change geometry in order to achieve some changes in visual appearance and, thus, communication behavior of the animation. Second, geometry changes might be necessary to adapt to the actual display capabilities. This will be explained in more detail. Finally, we will share the idea of a multicast scenario, where a number of different and distributed graphical workstation will display an animation.

Geometries and style of animation

Remember that the neutral face as well as each feature is described by a geometry. Obviously, we can change the whole set of geometries, without the need of re-authoring anything. Thus, we can use one animation and play it using a human face, but also e.g. a 'monster' face, dragon/horse/dog/whatever face. Imagine a large library of animations represented in our scheme - a designer, who wants to use his geometry of a face to play-back all these animations. All to do, would be to make the existing face smile, speak an 'A', and so on. But also changing only some of the geometries has interesting effects. One could change the style of some facial expressions, for instance the way a face smiles. This could be used to 'personalize' the facial expressions of a geometry.

Geometries and display capabilities

Of course, not every workstation displaying an animation will be the same. We would consider the geometries presented here neither very small, nor big. Remember that cost for displaying an animation increases linear with the number of vertices of the geometry. Therefore, small geometries will be displayed faster than large ones. On the other hand, large vertex counts might be desirable from a designer's point of view. Thus, we need to trade off between display limitations and designer's demand.

In recent years, simplification of polyhedral meshes has drawn much attention. One can find many schemes in the literature that allow a representation of meshes at different levels of accuracy. If the geometries would be represented at different levels of accuracy (and vertex count, of course) one could pick an appropriate resolution according to display capabilities and current workstation load.

Streaming animations to groups

Because of the very compact size of the animation itself, it can be communicated on demand or in real time from a server to clients. We envision multicast scenarios, where one server streams an animation to several clients. Contrary to most other approaches no quality of service considerations are necessary for the limited and varying bandwidth of communication connections. The amount of bandwidth needed for our representation will be always available, as long as the connection is not broken.

So, we can assume that every animation can be distributed among several clients. Every displaying workstation can adapt to its own limitations and to the user's needs. The former is obvious and was already explained in the previous section. For the latter, we think of needs in terms of usage, experience, and taste. The user might not want a fully detailed geometry, independent of the capabilities of the workstation. Or, younger users might want special characters to speak to them, while - at the same time - experienced users only need a very abstracted face that just conveys the information. In general, the independence of geometry and animation sequence is a key feature in distributed display scenarios.

7.2 Decomposing key frame animations

Assume we are given a scene comprising animated shapes described by key frame geometries F_i. All shapes are assumed to have a constant isomorphic topology K. We assume that all base shapes have vectors of same length and that the attributes are arranged identically. The state of an object in a key frame animation can then be calculated by interpolating between two consecutive key frames. Formally, this can be described as

$$A(t) = \sum_i a_i(t) F_i \tag{7.1}$$

where $A(t)$ stands for the object's state at time t and $a_i(t)$ are the weights describing the key frame interpolation, i.e.

$$a_i(t) = \left(0, \ldots, 0, \frac{t_{i+1} - t}{t_{i+i} - t_i}, \frac{t - t_i}{t_{i+i} - t_i}, 0, \ldots, 0\right) \tag{7.2}$$

where t_i is the time stamp of the i-th key frame.

However, if we want to separate geometry from animation an alternative representation would be useful, such as:

$$A(t) = a'_0(t) F'_0 + \sum_i a'_i(t) f'_i \tag{7.3}$$

where F'_0 represents the average geometry of the shape and the sum describes deviations from this representative geometry. This process of extracting a base component could be repeated in order to find the principal deviations from the average

geometry. Generally, it makes sense to sort the geometries based on their geometric importance. We denote this description as

$$A(t) = \sum_i \tilde{a}_i(t) B_i \tag{7.4}$$

where B_0 represents the average static geometry while the remaining factors B_i represent geometric changes with decreasing importance with respect to the reconstruction of the animation. Note that representations (7.1), (7.3), and (7.4) are just basis transformations of each other, i.e. a matrix multiplication transforms one into the other.

However, simple rigid motions of the object may render this approach obsolete because the linear deviations B_i from the base geometry B_0 cannot include e.g. rotations (see also Lengyel [1999]). For this reason we decompose the animation into rigid body motion and an elastic part first: First, all shapes are translated so that their center of mass coincides with the origin. Then, an affine map is computed minimizing the squared distance of corresponding vertices with regard to the first frame. The affine map is restricted to matrix representations with determinants greater zero, since reflections seem not appropriate and the matrix has to be invertible. Results of this approach are depicted in Figure 7.2.

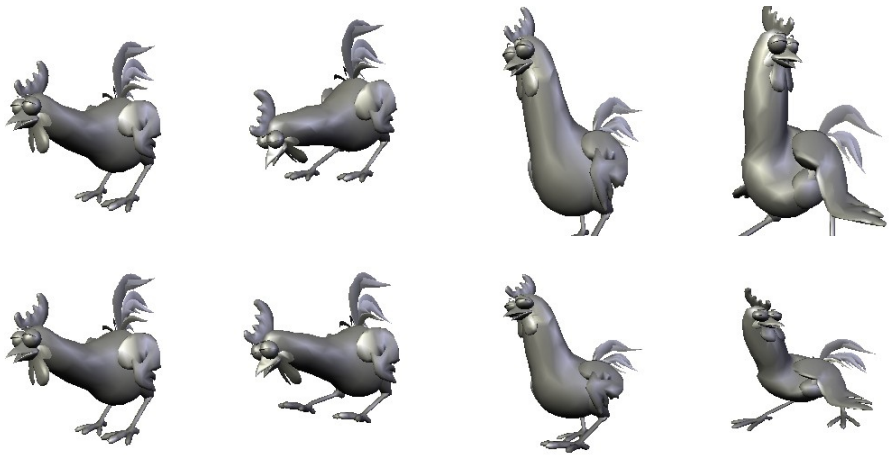


Figure 7.2: The normalization step. The upper row shows frames from a chicken animation translated so that the centers of mass coincide with the origin. The lower row shows transformed shapes where the squared distances of corresponding vertices are minimized.

If we assume our model to be represented in homogenous coordinates, we can write the necessary transformation as a single matrix multiplication. That is, instead of the key frames F_i we use transformed key-frames $T(t_i)F_i$. This has to be taken into account when the animation is reconstructed. Here, we have to use a

slightly different representation:

$$A(t) = T^{-1}(t) \sum_i \tilde{a}_i(t) B_i \tag{7.5}$$

Assumed the B_i behave as described above, we have a representation where animation and geometry are clearly decoupled. The geometry part is described mainly by B_0, while the main animation is described by the T_i and $\tilde{a}_i(t)$. B_1, B_2, B_3, and so on describe all possible deviations of the geometry.

This representation of the animation sequence makes it easy to perform compression and LOD operations. By restricting the representation to the first few components B_i, high compression ratios can be achieved while omitting only unimportant features. Furthermore, metric LOD techniques are directly supported since progressive meshes have to be generated and hold in memory for these few components only.

At the same time, the number of bases used in (7.4) affects the accuracy of the animation. Few B_1 result in a coarse representation of the animation, more B_i in higher accuracy. Thus, this representation is inherently progressive.

Further implications of this representation are shown and discussed in sections 5 and 6. In the upcoming section we explain how to find the above description.

7.2.1 Principal Component Analysis

A process of analyzing the relationship between base vectors of a space is the Principal Component Analysis (PCA). In our scenario, the PCA determines first the average shape that contains the common properties of the shapes in all key-frames. Other components will represent differences to this shape. This is also interesting when it comes to coding the bases of a shapes, as it exploits similarity and turns it into zeros, which are easily compressed using entropy encoding.

There are several ways of finding principal components. In our case we are not only concerned in finding the most important principal components to give a rough approximation of the shapes. In addition, we want to find an alternative basis and cut only a few non-contributing vectors. A way of finding this basis is the singular value decomposition (SVD) [Golub & Van Loan 1989].

Formally, we can write the non-rigid part of the original key-frames in matrix form:

$$F = (T_0 B_0, \ldots, T_{n-1} B_{n-1}) \tag{7.6}$$

Using the SVD we find the following:

$$F = BS\tilde{A} \tag{7.7}$$

The values of the diagonal matrix S are the singular values. The closer singular values are to zero, the closer the original base is to some linear dependencies. The first orthogonal matrix B contains the basis of the space with base vectors corresponding to the singular values, i.e. the rows contain the B_i we are searching for.

The last matrix $\tilde{A}$ contains the new weight vectors $\tilde{a}_i$. The matrix representation and SVD is visualized in Figure 7.3

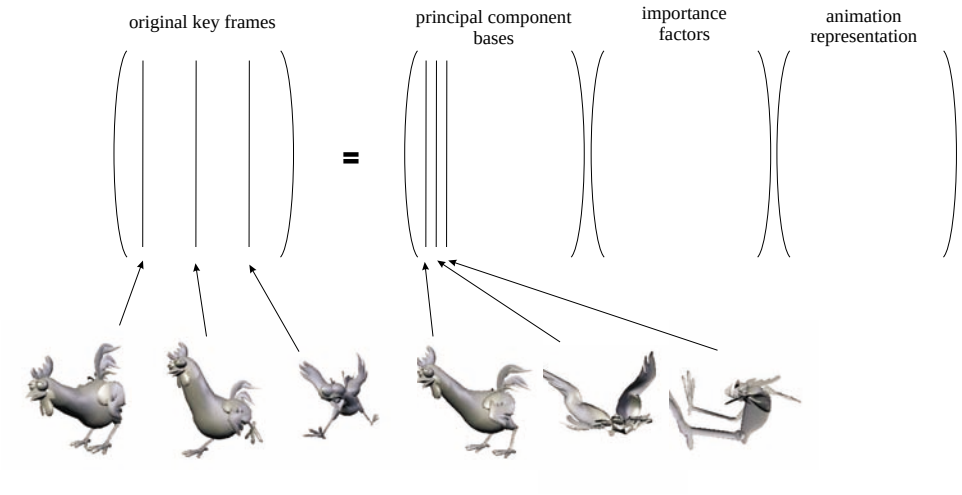


Figure 7.3: The Principal Component Analysis for geometric animations illustrated.

A severe problem with the SVD is that it is very costly and likely to reach its limits on modern computers when applied to matrices with the size of the vertex count of typical models times the numbers of key frames. A solution is to simplify the base shapes and to not consider every key frame. This is rectified by the spatial and temporal coherence typically exhibited in geometric animations. In particular, it is in many cases sufficient to consider only every second up to every fifth key frame. However, adaptive schemes for the selection of key frames would be desirable.

The representation vectors $\tilde{a}_i$ for key frame which have not been considered in the SVD can be obtained by projecting the key frame into the new basis. Since the basis constructed with the SVD is orthonormal, computing inner products of the key frames and the new base vectors is the desired projection.

7.2.2 Results

In this section we present results of computing a PCA for two animation sequences. We show how the PCA leads to a compressed progressive representation and fosters the exchange of geometry or behavior.

The first example is a part of the Chicken Crossing animation, in particular 400 frames of the chicken's geometry. The sequence is highly non-linear, i.e. it comprises rigid body motion and dynamic soft body changes. The geometry consists of 3030 vertices. Disregarding the topology information, this results in an uncompressed size of 400 frames $\times$ 3030 vertices $\times$ 3 dimensions $\times$ 4byte = 14.544.000

bytes.

To generate the principal component representation we first normalized the frames using the linear least squares fit. The resulting key-frames were composed into a 9090x400 matrix and a SVD was performed. The resulting orthogonal matrix was used to define the new base vectors replacing the key-frames.

Figure 7.4: Principal component analysis applied to the chicken animation. Prior to the SVD base shapes are normalized. The sequences above show frames 0, 80, 160, 240, 320, and 400 using different number of bases in the reconstruction.

We reconstructed the animated sequences using different numbers of base objects. This was done by setting appropriate singular values to zero. The results are shown in Figure 7.4. The animation still looks very reasonable with only 10 base shapes. Even the animation using only 5 bases can be used if viewed from far away. This would correspond to the definition of a complex LOD.

By omitting a number of base objects high compression ratios can be achieved. Table 7.1 shows the compression ratios for the animation. However, this animation is not typical. It comprises a highly deforming object. For typical animation

encoding	size	ratio
original	14,544,000	1
all base shapes	14,548,800	100.01 %
50 base shapes	1,818,600	12.50 %
10 base shapes	364,800	2.51 %
5 base shapes	181,8600	1.25 %
3 base shapes	109,116	0.75 %

Table 7.1: Compression ratios for principal component representation. Note that sizes and ratios include the additional costs for storing the transformation matrices from the normalization step.

sequences even better results can be expected.

In the second example we applied the principal component representation for facial animation. Here we used several facial expressions coded as polyhedral models with isomorphic vertex-edge topology. Key frames are generated by blending several expressions (e.g. speaking an “A” and smiling). In this system, the animation is represented as a vector over time that describes the linear combination of base shapes. Thus, animation representation and geometry are already decoupled in this specific application. In this example, we show how the geometry of the avatar can be exchanged with another geometry making use of the already existing animation descriptions.

We have used a feature based morphing technique to produce a mesh that can represent both, the original avatar and another face. By defining a few vertex-vertex correspondences we make sure that the same vertices represent common features in the source and the target model. This results in the convenient fact that we can combine the new face with expressions of the old one, e.g. we can add the smile defined in terms of the original avatar to the new face. This means, by defining the correspondences between the two neutral faces we get all other expressions of the face automatically. Play-back animations authored for the original avatar can directly be applied to the new face. In addition, we can morph between the two faces while the animation is performed since the avatar and the new face now share the same vertex edge-topology. This is an impressive example of the effectiveness of decoupling animation representation from geometry. Results are depicted in Figure 7.5.

Figure 7.5: Exchanging geometry in existing animations. a) A facial animation defined by a linear combination of base shapes. b) A featured-guided morph between the original avatar mesh and a new mesh. Topological merging is used to produce a mesh which represents both shapes. Feature control assures that the same vertices represent common features (e.g. mouth, eyes, etc.) c) The new mesh can be used with the existing animation with no additional user intervention. d) The morph can even be applied while the animation is performed.

7.3 Implementation in graphical standards

The concept of morphing among several base shapes has proven to be a universal tool in modeling and animation. We envision a construct that is able to perform the necessary operations as a part of modern graphics APIs. Inspired by a scenegraph we call the element *Morph Node* [Alexa et al. 2000].

The basic task a Morph Node should be able to accomplish is to interpolate among the vertex attributes of any number of homeomorphic shapes. Formally, a number of base shapes B_i are defined by different vertex attributes, V_i, where V_i may consist of a coordinate, a normal, color information, a texture coordinate, and possible other information. Given a vector $\mathbf{a}$, we need means to compute a shape $B(a)$, that is,

$$B(\mathbf{a}) = \sum_i a_i B_i = \left\{ \sum_i a_i V_i \right\} \tag{7.8}$$

However, it seems useful to be more flexible as to what is actually interpolated. For instance, the shapes might have color attributes specified, but color is the same for all base shapes. Obviously, it makes no sense to interpolate color. Or the user wants to fix a set of colors for the blended shape, independent of the colors actually set in the base shapes. This seems to be especially true for normals (see the upcoming section on Optimization issues).

Additionally, not all have to be interpolated linearly. A linear interpolation of normals is obviously not the only solution (to say the least). With respect to the general concept of a Morph Node one should be able to exclude specified attributes from the general linear interpolation and interpolate by means of other techniques.

To wrap up the needs for vertex attribute interpolation, we need to be able to

- interpolate linearly between any number of homeomorphic shapes,
- specify which subset of vertex attributes is actually interpolated
- specify fixed values attributes that are not interpolated or
- supply other interpolaters for these attributes.

For playing back animations we need an additional construct to interpolate the given key frame vectors over time. This could be done either linearly or by using splines. However, since the space requirements for a single key frame are so small, we can afford to specify as many key frames as are necessary for linear interpolation (e.g. more than 25 per second).

In the following subsections we will review the current 3D APIs with regard to the above stated requirements as well as propose ex- tensions and changes.

7.3.1 State-of-the-art

We would like to use web-based 3D application interfaces to play back geometric animations. We propose to use the techniques mentioned in the introduction to

accomplish this task. Thus, to playback an animation a workstation gets

1. the base shapes (several attributes, including position, normal, color, texture coordinate, etc.) and

2. a set of key-frames comprising a time stamp and a weight vector.

We will examine VRML-97, MPEG-4, Java3D, and X3D as to what they offer to play back an animation communicated in this way.

VRML-97

The first version of VRML, presented 1994, was largely a static scene description file format. The lack of animation and interaction was leading to VRML-2.0 which became the ISO standard VRML-97 [ISO JTC124 1997].

Even so, one of the main goals of the original VRML-2.0 proposal "Moving Worlds" was to create a specification which allows animation rich environments, the support for geometric morphing is limited. The designed concentrates on simple linear key-framed animation and supports only the linear interpolation of vertex position values of two shapes. The blending of more than two shapes is not supported at all. Furthermore, the `CoordinateInterpolater` only handles the vertex positions. Other vertex attributes like normal, color and texture coordinate are not handled by the `CoordinateInterpolator`. It is unusual to change the texture coordinate and color on vertex base during a key-frame animation and therefore not directly supported in VRML-97. The normal computation per vertex in contrast is very important but also time consuming and therefore different technics are provided by the specification:

1. The `NormalInterpolator` can be used to compute the normals for faces or vertex. Like the `CoordinateInterpolator`, the `Normalinterpolator` interpolates among a set of multi-valued `Vec3f` values. In addition, all output vectors will have been normalized by the node.

2. If the geometric node does not define any normals (the normal field is NULL), then the system automatically generates normals, using the given `normalPerVertex` and `creaseAngle` settings to determine if and how normals are smoothed across shared vertices.

3. If the base shape includes normal definitions, the fastest and simplest way is to keep the normals unchanged. Only the vertex positions are interpolated, which is not correct, but might be visually acceptable if the position changes are minimal

Beside `CoordinateInterpolator` and `NormalInterpolater` additional Interpolator for rotation and position are defined in the VRML-97 standard. These Interpolators are mainly used for rigid body animation which can be combined (e.g. swimming fish) with the key-frame description to get the desired effect.

There have already been various proposals to extend the animation capabilities of VRML-97. Two proposals lead to officially accepted WEB consortium working groups.

1. H-Anim: To create a standard VRML-97 representation for humanoids [Web3D Consortium 1999a].

2. Natural Language Processing (NLP): Will be used to allow natural language to interact with VRML-97 animation

Although both working groups are involved in animation description, none of these two work items is directly related to the work presented in this paper.

Java3D

Java3D is the API for 3D graphics within Java [Deering & Sowizral 1997]. It follows the scene graph architecture of VRML-97. A Shape in Java3D is represented by the node Shape3D comprising a Geometry and an Appearance node. Geometry is specified as an array of attributes, including coordinate, color (with or without alpha), normal, and texture coordinate. The appearance of a Shape can be controlled in various ways also depending on the geometry. In most application a material description will define the appearance.

Java3D has a node called “Morph”. This node accepts several geometries of exactly the same type, i.e. the same number of vertices with the same combination of attributes specified in the same way (indexed, stripped, etc.). One appearance object can be specified per Morph. The actual geometry can be altered by a call to the setWeights method.

In order to interpolate between key frames, Java3D offers Interpolators that are extended to take the time stamps of key frames into account.

As such, Java3D seems to offers the basic tools for the playback of animations given as base shapes and weight vectors. However, there are limitations in the design which hinder the use of Java3D in this context for now:

- base shapes need to have the same attributes

- and all these attributes are interpolated

- but no other than the predefined vertex attributes can be interpolated

We would like to give the user the flexibility to use one set of colors for all geometries without specifying them for every shape or, even worse, without interpolating them in any case. On the other hand it would be nice to allow the interpolation of materials.

In our example of facial animation the shapes have vertex colors, but these vertex colors do not change for different base shapes. In Java3D these float triples are interpolated whenever setWeights is called. Also, normals are interpolated whether

the user likes it or not (see the section on normal interpolation for other opportunities to set normals). Instead of interpolating only three floats for coordinates, Java3D interpolates nine and additionally normalizes the result of normal interpolation resulting in a more than three times worse performance.

MPEG-4

MPEG-4 represents one of the newest multimedia standards in the MPEG family. MPEG-4 supports the standard video encoding schemes well known from MPEG-2 based on efficient frame en- coding and motion compensation. In addition, MPEG-4 contains language components for 2d and 3d scene elements and scene structures based on VRML-97.

Specific extensions are related to the animation of artificial faces and bodies [Ostermann 1998]. The "facial animation object" in MPEG-4 can be used to render an animated face. The shape, texture, and expressions of the face can be defined by Facial Definition Parameters (FDPs) in BIFs. BIFs are downloadable models to configure a baseline face model or to install a specific face model at the beginning of a session along with the information about how to animate it. Facial Animation Parameters (FAPs) can then be used to animate this model.

Face Animation Table (FAT) within FDPs are used to perform the functional mapping from incoming FAPs to feature control points in the face mesh. Face Interpolation Technique (FIT) in BIFS allow the interpolation between different expressions based on weighted rational polynomial functions which is invoked by conditional evaluation of a Face Interpolation Graph. This functionality can be used for complex cross-coupling of FAPs to link their effects, or to interpolate FAPs missing in the stream using the FAPs that are available in the terminal. Face Animation in MPEG-4 provides for highly efficient coding of animation parameters that can drive an unlimited range of face models. The models themselves are not normative.

In addition, a new standard called "MPEG-4 The Body Animation" is being developed by MPEG in concert with the Humanoid Animation Working Group within the VRML Consortium, with the objective of achieving consistent conventions and control of body models which are being established by H-Anim. This Body Animation, to be standardized in MPEG-4 Version 2, is being designed into the MPEG-4 fabric to work in a thoroughly integrated way with face/head animation. Here, decoding and scene descriptions directly mirror to technology already proven in Face Animation and corresponding Body Definition Parameters (BDPs) and Body Animation Parameters (BAPs) exist.

The facial and body animation elements in VRML-97 represent a very interesting technology in the context of animated User Interfaces and User Interface Agents. The standardization of the control of face animations is one of the major achievements in this context. Technological systems supporting these concepts may, especially when based on FIT, lead to similar problems as stated in the motivation of this paper. The solutions provided in the following sections are therefore

also applicable in this context.

X3D

Another standardization activity in the area of 3d animation systems and exchange formats is X3D [Web3D Consortium 1999b]. The goal of X3D is to represent a next-generation extensible 3d graphics specification which makes use of the VRML-97 structure and expresses it by the means of XML: As such, in the current state of development X3D makes still use of the interpolator functionality known from VRML-97 for morphing.

7.3.2 Proposed extensions and changes

Since neither VRML nor Java3D provide means to implement the concept of a Morph Node as it would be suitable to support anima- tion and elaborate modeling techniques we propose the following changes and extensions to the existing standards.

Extensions to VRML-97

We propose extend VRML-97 by three nodes in order to accomplish the ideas of the Morph Node.

- A `VectorInterpolator` node, which handles interpolation of general float-type vectors.
- A `CoordinateMorpher`, which interpolates linearly among coordinate sets.
- A `NormalMorpher`, which interpolates linearly and normalizes normal sets.

In the following we give specification and details about these nodes.

```
VectorInterpolator {
        eventIn             SFFloat      set_fraction
        exposedField        MFFloat      key []
        exposedField        MFFloat      keyValue []
        eventOut            MFFloat      value_changed
}
```

The node is designed as an additional VRML-97 interpolator in the scene that defines a piecewise-linear function, $f(t)$, on the interval $[infty, \infty]$. The general aim of the node is the ability to interpolate between a set of vectors of any size. The node sends multiple-value results like the `CoordinateInterpolator` and `NormalInterpolator`. Therefore, the `keyValue` field is an $n \times m$ matrix of values, where n is the number of values in the key field and m the size of

the vector. The number of scalar values in the `keyValue` field shall be an integer multiple of the number of key-frames in the `key` field. That integer multiple defines only implicit the size of the vector and therefor the number of elements which will be contained in the `value_changed` events. The `value_changed` output is used in our proposal as input for the `set_weights` eventIn slot in the following `CoordinateMorpher` and `NormalMorpher` Node.

```
CoordinateMorpher {
        eventIn             MFFloat     set_weights
        exposedField        MFVec3f     keyValue []
        eventOut            MFVec3f     value_changed

}
```

The CoordinateMorpher node interpolates linearly among a set of `MFVec3f` values. Unlike the `CoordinateInterpolator` it does not interpolate two key frames but is able to blend any number of shapes. The number of coordinates in the `keyValue` shall be an integer multiple of the number of key-frames in the `key` field. That integer multiple defines how many coordinates will be contained in the `value_changed` eventout slot.

```
NormalMorpher {
        eventIn             MFFloat     set_weights
        exposedField        MFVec3f     keyValue []
        eventOut            MFVec3f     value_changed
}
```

The NormalMorpher node blends among a set of normal vector sets specified by the keyValue field. The output vector, `value_changed`, shall be a set of normalized vectors.

Changes in Java3D

The changes necessary to fit the existing Java3D MorphNode into our concept follow directly from the state-of-the-art section. In addition to what the Java3D API defines up to now we need ways to

1. specify the attributes which are actually interpolated,
2. supply sets of fixed attributes, and
3. supply other interpolators for specified attributes.

Furthermore, Java in general needs more accurate timing mechanisms to support synchronized playback of geometric animations.

Proposal for X3D

One of the main goals of the X3D proposal is to build the new technologies based on a small, lightweight core. Since the VRML-97 specification is considered to be large and complex to implement the base set will not include all elements of the predecessor. We propose to include the `CoordinateMorpher` instead of the `CoordinateInterpolator` into the core X3D specification. The `CoordinateMorpher` is a more general approach to shape animation and can substitute the Interpolators without any lose in functionality or performance.

7.3.3 Optimization issues

The general concept and implementation of a Morph Node is rather simple. However, a naive approach results in a complexity of O(nma), where n is the number of base shapes, m is the number vertices per base shape, and a this the number of attributes to be interpolated. That is, deciding which attributes really need to be interpolated is quite crucial. Especially normals are difficult to treat, since a simple linear interpolation does not yield correct results. We discuss ways to handle normals in the upcoming subsection.

Generally, base shapes might not differ in all the attributes for all vertices, and the system should exploit this fact as much as possible.

How to interpolate normals?

As said before, linear interpolation of normals might not be correct. First note that normals are not necessarily normalized after linear interpolation, which could cause shading to fail. Thus, we need to normalize the vectors after interpolation. But still, the result of linear interpolation seems to be not what one would expect. Instead we should use SLERP/quaternion interpolation. Unfortunately, even a quaternion based, correct direction interpolation of normals does not lead to the normals a linearly interpolated geometry actually has. For these diverse reasons we see the following for different ways to treat normals:

1. use only one set of normals (e.g. the normals of B0, or a set of mean normals among $B_0, \ldots, B_{n-1}$)
2. interpolate normals linearly and normalize
3. interpolate in quaternion space
4. compute new normals after geometry interpolation

While option one seems unacceptable at first sight it produces surprisingly good results. See also Figure 7.6, which compares an animation of swimming dolphin with correct normals (recalculate in each frame) with no normal interpolation (the corresponding movies can be downloaded from http://www.igd.fhg.de/ alexa/morphnode/).

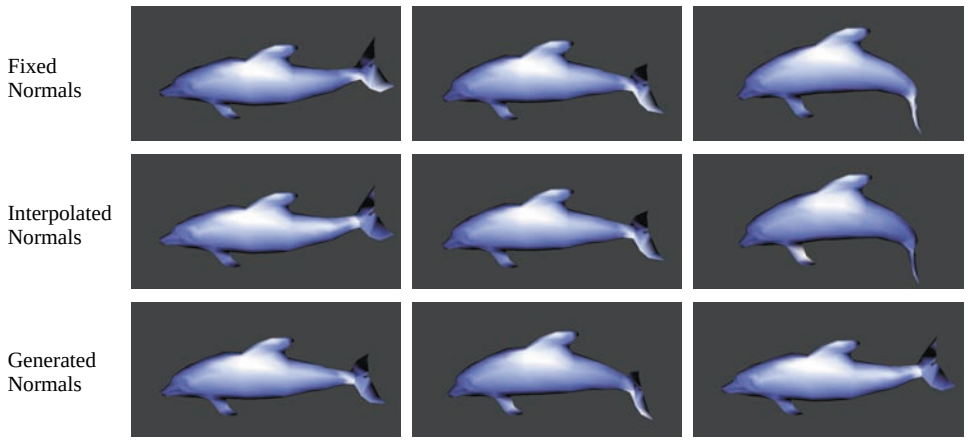


Figure 7.6: Comparison of normal strategies: The upper row shows different states of a dolphin animation with a fixed set of normals (i.e. the normals are the same in all three images). The middle row shows the same states with linearly interpolated and normalized normals. The normals in the lower row generated form the geometry for each frame.

	fixed	interpolated	generated
dolphin 1000 vertices	5.6 sec	6.8 sec	9.8 sec
pig-horse 14269 vertices	192.8 sec	245.7 sec	407.3 sec

Table 7.2: Times needed to render 1000 frames of an animation with different normal calculation strategies.

Of course, recalculation of normals might be necessary in a number of cases. We mainly considered linear interpolation and recalculation. Quaternion interpolation does not produce the correct normals of the interpolated shape yet is still more expensive than linear interpolation. A comparison of the remaining three approaches is given in table 1 below. Note that timings are heavily influenced by several other tasks the animation engine has to accomplish.

A more elaborated approach could compute geometry and normals asynchronously, i.e. updating normals only every f-th update of geometry. This would generate only slightly incorrect normals (which are acceptable, as was demonstrated before) and yet provide a high frame rate with respect to geometry.

Don't touch equal attributes

In many animations only parts of the attribute change. If we take a look at the facial animation example again we easily see that

- color - even though it is set - does not change at all for different base shapes

- vertex coordinates for several regions do not change or do not change much
- thus also the normals in theses regions do not change (much)

We have illustrated this for the facial animation example in Figure 7.7 (see also the corresponding animation at http://www.igd.fhg.de/ alexa/morphnode/). Here, we use the `ColorManipulator` node of the Avalon system [Behr 1999] to color code vertices depending on their velocity. Standard gray vertices mean fixed vertices, red vertices are moving. Note that only small number of vertices is turning red during the animation. Generally speaking, a subset of vertex attributes might

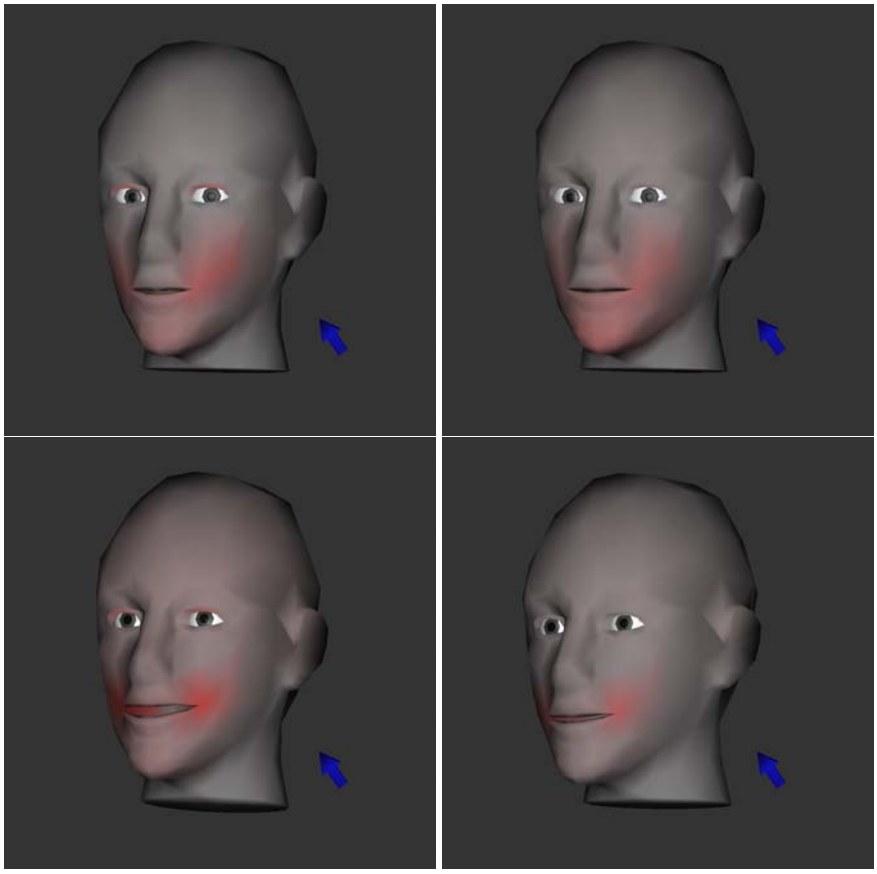


Figure 7.7: Color coding of moving vertices. Grey depicts static vertices, red color reflects high velocity.

be exactly or almost the same for all base shapes. The system should first iterate over all attributes for base shapes and identify similar attributes. This would result in information regarding which vertices and which attributes actually need to be interpolated.

A simple trick to exploit this information is to re-sort the vertices in each shape in a such a way that differing vertices appear first in the list/array. The loop nec-

essary for interpolation would only iterate over the first vertices not touching the fixed vertices.

Know what weights do to vertices

We can extend the concept of the last subsection to individual base shapes: some vertices might not change for a subset of base shapes. If the weights for the base shapes of this subset are all zero we do not need to recalculate these vertices. The basic problem is how to exploit the fact that some base shapes only need to interpolate a very small subset of vertices.

We propose the following solution. We reorder the vertices so that the following conditions describe the sequence of vertices:

1. None of the follwing cases.
2. $B_1 \neq B_0 = B_2 = B_3 = \ldots$
3. $B_2 \neq B_0 = B_1 = B_3 = \ldots$
4. and so on.

When looping over vertices we can include or exclude sets of vertices depending on the weight value of the respective base shape. For instance can we dismiss operations in block 2 if the weight corresponding to B_1 is set to zero.

7.4 Conclusions

We present an approach for representing animation sequences based on the principal components of key frame geometries. The principal component representation allows for an easy and adaptive lossy compression of animation sequences with factors up to 1% accepting loss in animation accuracy. It would be interesting to quantify that loss relative to the compression ratio. However, measures of geometric deformation seem to be topic of current research and not yet applicable. Moreover, standard compression techniques were not exploited at all in this calculation. Additional compression can be achieved by compressing the base shape matrices. Again, the principal component analysis reorganizes the base shapes so that bases with higher index will contain more zeros. This could be exploited with an additional entropy encoding.

The order of base objects naturally supports progressive transmission of the animation base shapes. At the same time, the inherent hierarchy could be used for LOD techniques. This could go hand in hand with a progressive mesh. The importance ordering of base shapes additionally eases mesh simplification since only the geometric features of a few meshes have to be taken into account for simplification. In first experiments we have found that standard simplification techniques can be extended to handle more than one mesh.

The animation itself is represented by small vectors if the number of necessary base shapes is small. Note that the number of necessary base shapes is bounded by the number of key frames. While the number of base shapes in our examples is much less than the number of key frames, it might be useful to break very long animation sequences into pieces. It would be interesting to exchange only parts of a base while streaming an animation.

Having small vectors representing the animation allows for streaming over virtually every network in real time. The decoupling of animation and geometry enables managing and changing animations, e.g. exchanging the animated object according to the client's display capabilities. Mapping existing animations to a new object offers new ways of authoring.

We also analyzed how to incorporate the necessary operations to play back animations defined as linear combinations of a small set of base shapes in state-of-the-art 3d graphics systems such as Java3D and VRML-97. Current shape blending methods based on interpolation nodes always perform an interpolation between the complete geometry of two base shapes and recalculate all object attributes such as normals and color anew. As a consequence, these systems lack performance when morphing is applied in real-time applications.

We propose a new morph node with enhanced flexibility and performance, supporting morphing between several objects and giving enhanced control over the morphing process. Most important, morphing calculations between attributes leading to no additional visual effect can be avoided in this way. For a number of test objects time we could prove a performance enhancement of upto 100%. Therefore, we recommend the consideration of our morph node in the ongoing X3D and

MPEG4 specification processes.

Chapter 8

Conclusions

In this work, mesh morphing techniques were analyzed, extended, and used to build linear spaces of three-dimensional shapes. The shape space has proven to be a useful concept to describe a shape in terms of other shapes. This notion is, indeed, intuitive for modeling (as shown for facial animations) and representing abstract data. The approach is particular strong for animated sequences of shapes. Such sequences were automatically converted into a representation in a linear space, allowing for efficient progressive streaming and storing of geometric animations.

However, we believe that representing shapes as compounds of other shapes is a universal concept with many more applications than we can even think of at this point.

8.1 Summary of contributions

To review, the primary contributions of this work as described in this dissertation are:

Mesh morphing Several contributions to the field of mesh morphing have been made, which are describe in the following.

Feature alignment A mesh morphing technique for topological spheres is introduced, which allows for particular easy feature specification. Features are specified as vertex-vertex correspondence. In contrast to other works, no minimum user interaction is necessary (such as in dissection-type methods) while allowing to specify as many features as wanted.

Mesh merging A new algorithm for generating one mesh from two input meshes (mesh overlay) is presented, generalizing the overlay problem to arbitrarily shaped objects. In contrast to other approaches, the algorithm is asymptotically optimal.

Local control In image morphing one can easily specify which region transforms when in the morph, e.g., first the nose, then the eyes and then

the rest of a face. We present similar techniques for mesh morphing. In particular Laplacian coordinates were introduce, representing mesh geometry in a differential way. This description is also useful for free-form deformation.

Vertex path A method to generate intuitive and well behaving vertex paths in morphing is introduced. The paradigm employed is that the object is transformed rigidly as much as possible to avoid unnecessary deformations. This is achieved by dissecting the shape into a simplicial complex and defining optimal simplex morph. With this approach also the interior of a shape is treated and not only the boundary.

Visualization of multiparameter data Morph spaces are shown to be a useful way to generate glyphs and icons in the visualization of multiparamter data. Additionally, the paradigm allows for a flexible and intuitive generation of the visualization.

Geometric animations Spaces of meshes are shown to be a particularly elegant and effective way to generate, store, and communicate geometric animations. The resulting compression is progressive and the achieved compression ratios progress over previous work. A preprocessing step first decouples rigid motion from the animation specification, thus, making the inherently linear process more effective.

8.2 Future research directions

This work has many ways to be extended or completed, partly because the idea of shape space has no limits in its application. The following subjects seem particular valuable or interesting:

Topology in mesh morphing Mesh morphing still suffers from differences in topology of the input. While some algorithms handle explicit topology changes, none can automatically deal with this problem. This is of paramount importance because of the nature of real world meshes (e.g., originating from range scanners): They contain some holes and tunnels, which may or may not be part of the model being represented. From a geometry point of view these small errors have negligible impact, however, mesh morphing algorithms will surely fail on this type of input.

It seems that this problem originates from the erroneous mesh. This is only partly true: At least two applications suffer from the topology problem in general.

1. Imagine a torus to be morphed into a large statue holding a small ring so that the statue has the same topology as the torus. The topologically correct mapping connects the tunnels in both models independent of their different geometric importance.

2. Imagine the statue to be morphed into another statue without the ring (i.e. a topological sphere). Why should the user be concerned with a very small feature?

Ideally, the user should be able to freely specify correspondence without any topological restrictions.

However, it might be that meshes are not suited for this application in general. We are currently analyzing how to use point set representations [Pfister et al. 2000; Malgouyres & Lenoir 2000] of shapes for morphing. The point set accurately describes the boundary of the sphere, however might be deformed into a point set representing a shape with a different topology.

Affinely independent representation For interpolation problems of all sorts intrinsic geometry representations seem useful. Ideally, the representation of geometry should be independent to affine transformations. We have presented Laplacian Coordinates, which are independent of translation and linear in the original coordinates. Linearity is important to ease numerical computations.

The basic idea of affinely independent coordinates is to represent each vertex as an affine sum of its neighbors. This representation is linear in the original coordinates and, by definition, insensitive to affine transforms. However, every subset (neighbors or a larger neighborhood) of vertices might be affinely dependent in $\mathbb{R}^3$ so that vertices cannot be represented as an affine combination in this subset.

It is interesting to characterize shapes that allow such an affine independent representation, i.e. where each sufficiently large subset of vertices is a base of $\mathbb{R}^3$. This representation has numerous applications in shape recognition, free-form deformation, or morphing.

Handling of attributes Even if coordinates are morphed in a pleasing way, it is not clear what to do with attributes such as normals. For example, if normals describe a crease in one model but the corresponding edge in the other model is smooth, when has the crease to disappear?

It seems worthwhile to give a precise answer to all these detail questions. Morphing, in the end, is a very esthetic subject and failure in the details leads to displeasing results.

Robust implementation Morphable meshes resulting from the merging process contain a lot of sliver triangles, which make them particularly prone to fail in further processing. Also small topological failures in the input cause most morphing algorithms to fail.

What is needed is a freely available, robust implementation of mesh morphing techniques. It seems that to date none of the publicized mesh morphing

techniques are implemented in public domain code. This would allow to experiment and improve the code using state-of-the-art geometric computation techniques such as arbitrary precision predicates.

Affine independence of shapes The approach presented here constructs spaces of meshes with one consistent topology, where the base meshes are described by vertex vector. Compound shapes are just linear combinations of the vertex vectors. For many applications it is necessary to have an affinely independent basis. This can be computed on the basis of the vertex vectors in our approach. However, the affine independence of vertex vectors does not guarantee the affine independence of shapes. Imagine a square with one interior vertex connected to the boundary vertices. Wherever the interior vertex is located the shape is the same. Yet, two different vertex configurations are affinely independent.

It would be interesting to code geometry in such a way that different affine independent geometry descriptors imply affine independent shapes. It seems the problem of absolute coordinates lies in the possible redundancy: If redundant information is stored a degree of freedom is added in the representation, which could lead to independent description of dependent shapes. This implies that topology should be exploited when defining geometry. However, the question how to do this is open.

Shape registration Eigenfaces are an early example of using a linear space for recognition of objects. We would like to see spaces of meshes to be used for recognition. Together with the above mentioned affinely independent description of geometry they represent a good candidate space which is easy to search.

Feature-oriented SVD The SVD used to compute the basis for a shape space so far takes into account only geometry. Sometimes, however, small geometrical features are of special importance and should get an extra weight. An example could be a human figure standing on the floor: The contact of human and floor is of special importance because the human protruding the floor or floating are recognized easily by a human observer.

An idea could be to deform the geometric space so that regions of particular interest get enlarged. Then a regular SVD could be used. In the example above, the space around the wall would be enlarged, so that differences in this area get larger.

References

ALEXA, M. 1999. Merging polyhedral shapes with scattered features. In *Proceedings of the International Conference on Shape Modeling and Applications (SMI-99)*, B. Werner, Ed. IEEE Computer Society, Los Alamitos, CA, 202–210.

ALEXA, M. 2000. Merging polyhedral shapes with scattered features. *The Visual Computer 16*, 1, 26–37. ISSN 0178-2789.

ALEXA, M. 2001a. Local control for mesh morphing. In *Proceedings of the International Conference on Shape Modeling and Applications (SMI-01)*, B. Werner, Ed. IEEE Computer Society, Los Alamitos, CA, 209–215.

ALEXA, M. 2001b. Mesh morphing star. *Eurographics 2001 State of The Art Reports*, 1–20. ISSN 1017-4656.

ALEXA, M. 2002a. Differential coordinates for mesh morphing and deformation. *The Visual Computer 18*. to be published.

ALEXA, M. 2002b. Recent advances in mesh morphing. *Computer Graphics Forum 21*, 2. to be published.

ALEXA, M., BEHR, J., AND MÜLLER, W. 2000. The morph node. *Web3D - VRML 2000 Proceedings*, 29–34. ISBN 1-58113-211-5.

ALEXA, M., BERNER, U., HELLENSCHMIDT, M., AND RIEGER, T. 2001. An animation system for user interface agents. In *WSCG 2001 Conference Proceedings*, V. Skala, Ed.

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. *Proceedings of SIGGRAPH 2000*, 157–164. ISBN 1-58113-208-5.

ALEXA, M. AND MÜLLER, W. 1998a. The morphing space. GRIS-98-2, Technische Universität Darmstadt.

ALEXA, M. AND MÜLLER, W. 1998b. Visualization by metamorphosis. *Visualization '98 Late Breaking Hot Topics*.

ALEXA, M. AND MÜLLER, W. 1999a. The morphing space. *Seventh International Conference in Central Europe on Computer Graphics and Visualization (Winter School on Computer Graphics)*. ISBN 80-7082-490-5. Held in University of West Bohemia, Plzen, Czech Republic, 10-14 February 1999.

ALEXA, M. AND MÜLLER, W. 1999b. Visualization by examples: Mapping data to visual representations using few correspondences. *Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization*.

ALEXA, M. AND MÜLLER, W. 2000. Representing animations by principal components. *Computer Graphics Forum 19*, 3 (August), 411–418. ISSN 1067-7055.

ARAD, N. AND REISFELD, D. 1994. Image warping using few anchor points and radial functions. *Computer Graphics Forum 14*, 1 (January), 35–46.

ARONOV, B., SEIDEL, R., AND SOUVAINE, D. 1993. On compatible triangulations of simple polygons. *Computational Geometry: Theory and Applications 3*, 27–35.

BAO, H. AND PENG, Q. 1998. Interactive 3d morphing. *Computer Graphics Forum 17*, 3, 23–30. ISSN 1067-7055.

BEDDOW, J. 1990. Shape coding of multidimensional data on a microcomputer display. In *Proceedings of Visualization 90*. 238–246.

BEHR, J. 1999. AVALON - A VR/AR system using VRML as application description language. http://www.zgdv.de/
avalon.

BERTIN, J. 1983. *Semiology of Graphics*. The University of Wisconsin Press.

BLANZ, V. AND VETTER, T. 1999. A morphable model for the synthesis of 3d faces. *Proceedings of SIGGRAPH 99*, 187–194. ISBN 0-20148-560-5. Held in Los Angeles, California.

BOYER, R. AND SAVAGEAU, D. 1985. *Places Rated Almanac*. Rand McNally.

BRENT, R. P. 1973. *Algorithms for Minimization without Derivatives*. Prentice-Hall, Englewood Cliffs, N.J.

BRODLIE, K. W. 1993. A classification scheme for scientific visualisation. In *Animation and Scientific Visualisation: tools and applications*, R. A. Earnshaw and D. Watson, Eds. Academic Press, 125–140.

BUJA, A., MCDONALD, J. A., MICHALAK, J., AND STUETZLE, W. 1991. Interactive data visualization using focusing and linking. *Visualization '91*, 156–163.

CHAZELLE, B. 1990. Triangulating a simple polygon in linear time. *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, 220–230.

CHAZELLE, B. AND PALIOS, L. 1989. Triangulating a non-convex polytope. In *Proceedings of the 5th Annual Symposium on Computational Geometry (SCG '89)*, K. Mehlhorn, Ed. ACM Press, Saarbrücken, FRG, 393.

CHENG, H., EDELSBRUNNER, H., AND FU, P. 1998. Shape space from deformation. *Pacific Graphics '98*. Held in Singapore.

CHERNOFF, H. 1973. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association 68*, 361–368.

CLEVELAND, W. S. 1985. *The Elements of Graphing Data*. Wadsworth. ISBN 0-534-03730-5.

COHEN, S., ELBER, G., AND BAR-YEHUDA, R. 1997. Matching of freeform curves. *Computer-aided Design 29,* 5, 369–378.

COHEN-OR, D. AND CARMEL, E. 1998. Warp-guided object-space morphing. *The Visual Computer 13,* 9-10, 465–478. ISSN 0178-2789.

COHEN-OR, D., SOLOMOVICI, A., AND LEVIN, D. 1998. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics 17,* 2 (April), 116–141. ISSN 0730-0301.

DE BERG, M., VAN KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O. 1997. *Computational Geometry – Algorithms and Applications*. Springer-Verlag, Berlin Heidelberg.

DECARLO, D. AND GALLIER, J. 1996. Topological evolution of surfaces. *Graphics Interface '96,* 194–203. ISBN 0-9695338-5-3.

DEERING, M. AND SOWIZRAL, H. 1997. *Java3D Specification, Version 1.0*. Sun Microsystems, 2550 Garcia Avenue, Mountain View, CA 94043, USA.

DÖRNER, R., LUCKAS, V., AND SPIERLING, U. 1997. Ubiquitous animation - an element-based concept to make 3d animations commonplace. In *Viusal Procedings SIGGRAPH '97*. ACM Press.

DYN, N. 1989. Interpolation and approximation by radial and related functions. In *Approximation Theory VI*. Vol. 1. Academic Press, 211–234.

ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution analysis of arbitrary meshes. *Proceedings of SIGGRAPH 95,* 173–182. ISBN 0-201-84776-0. Held in Los Angeles, California.

ECKSTEIN, I., SURAZHSKY, V., AND GOTSMAN, C. 2001. Texture mapping with hard constraints. *Computer Graphics Forum 20,* 3, 95–104. ISSN 1067-7055.

EDELSBRUNNER, H. 1999. Deformable smooth surface design. *Discrete and Computational Geometry 21,* 1 (Jan.), 87–115.

EKMAN, P. AND FRIESEN, W. V. 1978. *Facial Action Coding System (Investigator's Guide)*. Consulting Psychologists Press, Inc., Palo Alto, California, USA.

FINKE, U. AND HINRICHS, A. 1995. Overlaying simply connected planar subdivisions in linear time. In *Proceedings of the 11th Annual Symposium on Computational Geometry*. ACM Press, New York, NY, USA, 119–126.

FLOATER, M. S. 1997. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design 14,* 3, 231–250. ISSN 0167-8396.

FLOATER, M. S. 2001. Convex combination maps. *Algorithms for Approximation IV*.

FLOATER, M. S. 2002. One-to-one piecewise linear mappings over triangulations. *Math. Comp. to appear*.

FLOATER, M. S. AND GOTSMAN, C. 1999. How to morph tilings injectively. *Journal of Computational and Applied Mathematics 101,* 117–129.

FREITAG, L., JONES, M., AND PLASSMANN, P. 1999. A parallel algorithm for mesh smoothing. *SIAM Journal on Scientific Computing 20,* 6 (Nov.), 2023–2040.

FUJIMURA, K. AND MAKAROV, M. 1998. Folder-free image warping. *Graphical Models and Image Processing 60,* 2 (March), 100–111.

GARLAND, M. AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. *Proceedings of SIGGRAPH 97*, 209–216. ISBN 0-89791-896-7. Held in Los Angeles, California.

GOLUB, G. H. AND VAN LOAN, C. F. 1989. *Matrix Computations*, Second ed. Johns Hopkins Series in the Mathematical Sciences, vol. 3. The Johns Hopkins University Press, Baltimore, MD, USA. Second edition.

GOTSMAN, C. AND SURAZHSKY, V. 2001. Guaranteed intersection-free polygon morphing. *Computers & Graphics 25,* 1 (February), 67–75. ISSN 0097-8493.

GREGORY, A., STATE, A., LIN, M., MANOCHA, D., AND LIVINGSTON, M. 1998. Feature-based surface decomposition for correspondence and morphing between polyhedra. *Computer Animation '98*. Held in Philadelphia, Pennsylvania, USA.

GREGORY, A., STATE, A., LIN, M. C., MANOCHA, D., AND LIVINGSTON, M. A. 1999. Interactive surface decomposition for polyhedral morphing. *The Visual Computer 15,* 9, 453–470. ISSN 0178-2789.

GUMHOLD, S. 2000. Personal communication on embedding meshes.

HOPPE, H. 1996. Progressive meshes. *Proceedings of SIGGRAPH 96*, 99–108. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

HORMANN, K. AND GREINER, G. 2000. Mips: an efficient global parametrization method. In *Curve and Surface Design: Saint-Malo 1999*, P. S. P.-J. Laurent and L. L. Schumaker, Eds. Vanderbilt University Press, 153–162.

HORMANN, K., GREINER, G., AND CAMPAGNA, S. 1999. Hierarchical parametrization of triangulated surfaces. In *Vision, Modeling and Visualization '99*, B. Girod, H. Niemann, and H.-P. Seidel, Eds. infix, 219–226.

HUBELI, A. AND GROSS, M. 2001. Multiresolution feature extraction for unstructured meshes. In *IEEE Visualization 2001*. 287–294. ISBN 0-7803-7200-x.

ISO JTC124. 1997. VRML-97. ISO/IEC 14772-1.

ISO JTC/WG11. 1997. MPEG 4. ISO/IEC 14496-1, ISO/IEC 14496-2.

KANAI, T. AND SUZUKI, H. 2000. Approximate shortest path on a polyhedral surface based on selective refinement of the discrete graph and its applications. *Proc. Geometric Modeling and Processing 2000*, 241–250.

KANAI, T., SUZUKI, H., AND KIMURA, F. 1997. 3d geometric metamorphosis based on harmonic map. *Pacific Graphics '97*. Held in Seoul, Korea.

KANAI, T., SUZUKI, H., AND KIMURA, F. 1998. Three-dimensional geometric metamorphosis based on harmonic maps. *The Visual Computer 14*, 4, 166–176. ISSN 0178-2789.

KANAI, T., SUZUKI, H., AND KIMURA, F. 2000. Metamorphosis of arbitrary triangular meshes. *IEEE Computer Graphics & Applications 20*, 2 (March/April), 62–75. ISSN 0272-1716.

KARNI, Z. AND GOTSMAN, C. 2000. Spectral compression of mesh geometry. *Proceedings of SIGGRAPH 2000*, 279–286. ISBN 1-58113-208-5.

KARYPIS, G. AND KUMAR, V. 1998. Multilevel k-way hypergraph partitioning. Tech. Rep. 98–036, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455.

KENT, J., PARENT, R., AND CARLSON, W. E. 1991. Establishing correspondences by topological merging: A new approach to 3-d shape transformation. *Graphics Interface '91*, 271–278.

KENT, J. R., CARLSON, W. E., AND PARENT, R. E. 1992. Shape transformation for polyhedral objects. *Computer Graphics (Proceedings of SIGGRAPH 92) 26*, 2 (July), 47–54. ISBN 0-201-51585-7. Held in Chicago, Illinois.

KIRBY, M. AND SIROVICH, L. 1990. Application of the karjunen-loeve procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*, 1, 103–108.

KIRKPATRICK, S., GELLATT, C. D., AND VECCHI, M. P. 1983. Simulated annealing. *Science 220*, 671.

KLEIN, R. 1998. Multiresolution representations for surfaces meshes based on the vertex decimation method. *Computers & Graphics 22*, 1 (February), 13–26. ISSN 0097-8493.

KOBBELT, L. 2000. sqrt(3) subdivision. *Proceedings of SIGGRAPH 2000*, 103–112. ISBN 1-58113-208-5.

KOBBELT, L., CAMPAGNA, S., AND SEIDEL, H.-P. 1998. A general framework for mesh decimation. *Graphics Interface '98*, 43–50. ISBN 0-9695338-6-1.

LAZARUS, F. AND VERROUST, A. 1997. Metamorphosis of cylinder-like objects. *The Journal of Visualization and Computer Animation 8*, 3, 131–146. ISSN 1049-8907.

LAZARUS, F. AND VERROUST, A. 1998. Three-dimensional metamorphosis: a survey. *The Visual Computer 14*, 8-9, 373–389. ISSN 0178-2789.

LEE, A., DOBKIN, D., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution mesh morphing. *Proceedings of SIGGRAPH 99*, 343–350. ISBN 0-20148-560-5. Held in Los Angeles, California.

LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. 1998. Maps: Multiresolution adaptive parameterization of surfaces. *Proceedings of SIGGRAPH 98*, 95–104. ISBN 0-89791-999-8. Held in Orlando, Florida.

LEE, S.-Y., CHWA, K.-Y., SHIN, S. Y., AND WOLBERG, G. 1995. Image metamorphosis using snakes and free-form deformations. *Proceedings of SIGGRAPH 95*, 439–448. ISBN 0-201-84776-0. Held in Los Angeles, California.

LENGYEL, J. E. 1999. Compression of time-dependent geometry. *1999 ACM Symposium on Interactive 3D Graphics*, 89–96. ISBN 1-58113-082-1.

LÉVY, B. 2001. Constrained texture mapping for polygonal meshes. In *Proceedings of ACM SIGGRAPH 2001*. Computer Graphics Proceedings, Annual Conference Series. ACM Press / ACM SIGGRAPH, 417–424. ISBN 1-58113-292-1.

LÉVY, B. AND MALLET, J.-L. 1998. Non-distorted texture mapping for sheared triangulated meshes. *Proceedings of SIGGRAPH 98*, 343–352. ISBN 0-89791-999-8. Held in Orlando, Florida.

LINDSTROM, P. AND TURK, G. 1998. Fast and memory efficient polygonal simplification. *IEEE Visualization '98*, 279–286. ISBN 0-8186-9176-X.

LOOP, C. AND DEROSE, T. 1990. Generalized b-spline surfaces of arbitrary topology. *Computer Graphics (Proceedings of SIGGRAPH 90) 24*, 4 (August), 347–356. ISBN 0-201-50933-4. Held in Dallas, Texas.

MAGNENAT-THALMANN, N., PRIMEAU, E., AND THALMANN, D. 1988. Abstract muscle action procedures for human face animation. *The Visual Computer 3*, 5, 290–297.

MALGOUYRES, R. AND LENOIR, A. 2000. Topology preservation within digital surfaces. *Graphical Models 62*, 2 (March), 71–84. ISSN 1524-0703.

MICHIKAWA, T., KANAI, T., FUJITA, M., AND CHIYOKURA, H. 2001. Multiresolution interpolation meshes. In *9th Pacific Conference on Computer Graphics and Applications*. IEEE, 60–69. ISBN 0-7695-1227-5.

MULLER, D. E. AND PREPARATA, F. P. 1978. Finding the intersection of two convex polyhedra. *Theoretical Computer Science 7*, 2, 217–236.

MÜLLER, W. AND ALEXA, M. 1998. Using morphing for information visualization. *Workshop on New Paradigms in Information Visualization and Manipulation (NPIV '98)*, 76–79.

MÜLLER, W., ALEXA, M., RIEGER, T., AND BRAUN, N. 2000. Ein flexibles Präsentationssystem für User-Interface-Agenten. *Workshop Digital Storytelling (DISTEL)*, 163–175. ISBN 3-8167-5566-6.

NGO, T., CUTRELL, D., DANA, J., DONALD, B., LOEB, L., AND ZHU, S. 2000. Accessible animation and customizable graphics via simplicial configuration modeling. *Proceedings of SIGGRAPH 2000*, 403–410. ISBN 1-58113-208-5.

OSTERMANN, J. 1998. Animation of synthetic faces in mpeg-4. *Computer Animation '98*. Held in Philadelphia, Pennsylvania, USA.

PARKE, F. I. 1979. Computer graphic models for the human face. *Proc. COMPSAC, The IEEE Computer Society's Third International Computer Software and Applications Conference*.

PARKE, F. I. 1982. Parameterized models for facial animation. *IEEE Computer Graphics & Applications 2*, 61–68.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: Surface elements as rendering primitives. *Proceedings of SIGGRAPH 2000*, 335–342. ISBN 1-58113-208-5.

PICKETT, R. M. AND GRINSTEIN, G. G. 1988. Iconographics display for visualizing multidimensional data. *Proceedings of IEEE Conference on Systems, Man, and Cybernetics*, 514–519.

PINKALL, U. AND POLTHIER, K. 1993. Computing discrete minimal surfaces and their conjugates. *Experimanetal Mathematics 2*, 1, 15–36.

POLTHIER, K. 2000. Conjugate harmonic maps and minimal surfaces. Tech. Rep. Preprint No. 446, TU Berlin, SFB 288.

PRAUN, E., SWELDENS, W., AND SCHRÖDER, P. 2001. Consistent mesh parameterizations. *Proceedings of SIGGRAPH 2001*, 179–184. ISBN 1-58113-292-1.

PREPARATA, F. P. AND SHAMOS, M. I. 1985. *Computational Geometry: An Introduction*. Texts and Monographs in Computer Science. Springer-Verlag, Berlin, Germany.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 1992. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*. Cambridge University Press, Cambridge. ISBN 0-521-43108-5.

RHEINGANS, P. 1992. Color, change, and control for quantitative data display. In *Visualization '92*. 252–259.

RUPRECHT, D. AND MULLER, H. 1995. Image warping with scattered data interpolation. *IEEE Computer Graphics & Applications 15,* 2 (March), 37–43.

SCHROEDER, W. J. 1997. A topology modifying progressive decimation algorithm. *IEEE Visualization '97*, 205–212. ISBN 0-58113-011-2.

SCHROEDER, W. J., ZARGE, J. A., AND LORENSEN, W. E. 1992. Decimation of triangle meshes. *Computer Graphics (Proceedings of SIGGRAPH 92) 26,* 2 (July), 65–70. ISBN 0-201-51585-7. Held in Chicago, Illinois.

SEDERBERG, T. W., GAO, P., WANG, G., AND MU, H. 1993. 2d shape blending: An intrinsic solution to the vertex path problem. *Proceedings of SIGGRAPH 93*, 15–18. ISBN 0-201-58889-7. Held in Anaheim, California.

SEDERBERG, T. W. AND GREENWOOD, E. 1992. A physically based approach to 2d shape blending. *Computer Graphics (Proceedings of SIGGRAPH 92) 26,* 2 (July), 25–34. ISBN 0-201-51585-7. Held in Chicago, Illinois.

SHAPIRA, M. AND RAPPOPORT, A. 1995. Shape blending using the star-skeleton representation. *IEEE Computer Graphics & Applications 15,* 2 (March), 44–50.

SHAPIRO, A. AND TAL, A. 1998. Polyhedron realization for shape transformation. *The Visual Computer 14,* 8-9, 429–444. ISSN 0178-2789.

SHOEMAKE, K. AND DUFF, T. 1992. Matrix animation and polar decomposition. *Graphics Interface '92*, 258–264.

SPANIER, E. H. 1966. *Algebraic Topology*. McGraw-Hill, New York.

SUN, Y. M., WANG, W., AND CHIN, F. Y. L. 1997. Interpolating polyhedral models using intrinsic shape parameters. *The Journal of Visualization and Computer Animation 8,* 2 (April-June), 81–96. ISSN 1049-8907.

SURAZHSKY, T. AND ELBER, G. 2001. Matching free form surfaces. *Computers & Graphics 26,* 1, ??–?? ISSN 0097-8493.

TAL, A. AND ELBER, G. 1999. Image morphing with feature preserving texture. *Computer Graphics Forum 18,* 3 (September), 339–348. ISSN 1067-7055.

TAUBIN, G. 1995. A signal processing approach to fair surface design. *Proceedings of SIGGRAPH 95*, 351–358. ISBN 0-201-84776-0. Held in Los Angeles, California.

TUFTE, E. R. 1983. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut.

TURK, M. AND PENTLAND, A. 1991. Eigenfaces for recognition. *Journal of Cognitive Neuro Science 3,* 1, 71–86.

TUTTE, W. T. 1963. How to draw a graph. *Proc. London Mathematical Society 13*, 743–768.

WATERS, K. 1987. A muscle model for animating three-dimensional facial expression. *Computer Graphics (Proceedings of SIGGRAPH 87) 21*, 4 (July), 17–24. Held in Anaheim, California.

WEB3D CONSORTIUM. 1999a. H-Anim. *http://ece.uwaterloo.ca:80/˜h-anim*.

WEB3D CONSORTIUM. 1999b. X3D. http://www.web3d.org/x3d.

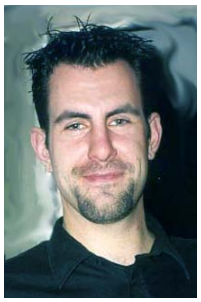
WOLBERG, G. 1998. Image morphing: a survey. *The Visual Computer 14*, 8-9, 360–372. ISSN 0178-2789.

ZHANG, Y. 1996. A fuzzy approach to digital image warping. *IEEE Computer Graphics & Applications 16*, 4 (July), 34–41. ISSN 0272-1716.

ZIGELMANN, G., KIMMEL, R., AND KIRYATI, N. 2002. Texture mapping using surface flattening vie multi-dimensional scaling. *IEEE Transactions on Visualization and Computer Graphics to appear*.

ZÖCKLER, M., STALLING, D., AND HEGE, H.-C. 2000. Fast and intuitive generation of geometric shape transitions. *The Visual Computer 16*, 5, 241–253. ISSN 0178-2789.

Marc Alexa



leads the project group *3D graphics computing* within GRIS, Technische Universitt Darmstadt, Germany. He received his MS degree in Computer Science with honors from TU Darmstadt. His research interests include shape modeling, transformation, and animation as well as conversational user interfaces and information visualization.