



informatics mathématiques
Inria

Algorithms and techniques for virtual camera control

Session 1: Introduction

M. Christie, Univ. Rennes 1
C. Lino, Univ. Rennes 1
R. Ranon, Univ. Udine

Presenters



Marc Christie - Associate Professor, University of Rennes 1, France - marc.christie@irisa.fr



Christophe Lino - Postdoctoral Fellow, University of Rennes 1, France - christophe.lino@irisa.fr



Roberto Ranon - Assistant Professor, University of Udine, Italy – roberto.ranon@uniud.it



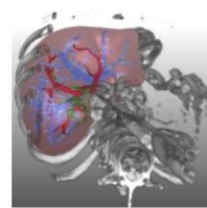
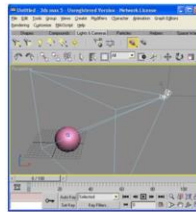
Tutorial Outline

Basic Knowledge	10 mn	Christophe Lino
User-Controlled Cameras	30 mn	Marc Christie
Computer-controlled cameras part 1 - Viewpoint Computation	30 mn	Roberto Ranon
Computer-controlled cameras part 2 - Camera Motions	30 mn	Marc Christie
Automated Editing	30 mn	Christophe Lino
Applications, trends and issues	30 mn	all



What is virtual camera control?

- process by which the camera is interactively or automatically controlled in a 3D environment
 - e.g. games, virtual storytelling, modeling, data / scientific visualisation
- encompasses a collection of techniques to:
 - aid the user in controlling the camera
 - place the camera in a suitable position
 - maintain the visibility of targets
 - make well-composed shots
 - plan camera paths
 - perform cuts between shots



Application: Games

- require camera control
 - during game play (real-time)
 - between game play (cut scenes)
- available resources tightly constrained
- classes of viewpoint
 - first person
 - third person
 - bird's eye
 - cinematic
- key problems:
 - occlusion vs geometric complexity
 - gameplay vs cinematic qualities
 - visual consistency



Interactive computer games serve the benchmark application for camera control techniques. Most importantly, they impose the necessity for real-time camera control. A canonical camera control problem involves following one or more characters whilst simultaneously avoiding occlusions in a highly cluttered environment. Furthermore, narrative aspects of real-time games can be supported by judicious choice of shot edits both during and between periods of actual game play. The increasing geometric complexity of games means that most deployed camera control algorithms in real-time 3D games rely upon fast (but fundamentally limited) visibility checking techniques.

Camera control in games has received considerably less attention in computer games than visual realism, though as John Giors (a game developer at Pandemic Studios) noted, “the camera is the window through which the player interacts with the simulated world”. Recent console game releases demonstrate an increasing

desire to enhance the portrayal of narrative aspects of games and furnish players with a more cinematic experience. This requires the operationalization of the rules and conventions of cinematography. This is particularly relevant in the case of games that are produced as a film spin-offs, where mirroring the choices of the director is an important means of relating the game play to the original cinematic experience.

Example: Heavy Rain



Heavy Rain, © Quantic Dream, 2010



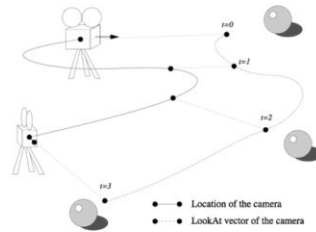
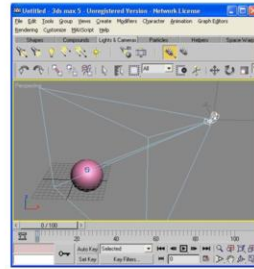
This video can be found at

<https://www.youtube.com/watch?v=fMK6sTnMxBI>.

Heavy Rain, as other similar games that are being increasingly developed, closely mimic the language of movies in presenting the virtual experience to the user, using editing and camera movements that follow a specific cinematographic style. However, all cameras in the game have been more or less manually designed for the range of actions and events that the game can display (and that have to be thus known in advance).

Application: Modelers

- 3D artists specify:
 - camera position
 - look-at / up vectors
- control provided:
 - classical interpolation methods (splines with key frames/control points)
 - fine control of the velocity curves supported
 - target constraints supported
- other basic notions from cinematic practice are not supported (e.g. framing)
- designer is the cameraman (not the director)



In three-dimensional modeling environments, virtual cameras are typically configured through the specification of the location of the camera and two vectors that represent the look-at and up directions of the camera. The specification of camera motion is usually undertaken through a combination of direct editing and interpolation, such as the use of splines with key frames and/or control points. Animation of the camera is realized by interpolating the camera location, up and look-at vectors across key frames. Fine control of camera speed is provided through the ability to manipulate the velocity graphs for each curve.

A set of complementary tools provides modelers with the ability to use the position of a unique static or dynamic target object to constrain the look-at vector. Modelers may also allow the use of offset parameters to shift the camera a small amount from the targeted object or path. Similarly, some tools allow constraints to be added to fix each component of the look-at vector individually.

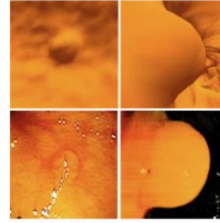
Physical metaphors are also used to aid tracking, such as virtual rods that link the camera to a target object. With the possibility to extend the functionality of modelers through scripting languages and plug-ins, new controllers for cameras can be readily implemented (e.g. using physics-based systems). Furthermore, with the rise of image-based rendering, the creation of camera paths using imported sensor data from real cameras is increasingly popular.

In practice, the underlying camera control model (i.e. two spline curves) is not well suited to describing the behavioral characteristics of a real world cameraman, or the mechanical properties of real camera systems. Despite the fact that a number of proposals exist for describing cinematic practice in terms of camera position, orientation and movement, most modelers have not attempted to explicitly incorporate such notions in their tools. Even basic functionality, such as automatically moving to an unoccluded view of a focal object, cannot be found in current commercial modeling environments.

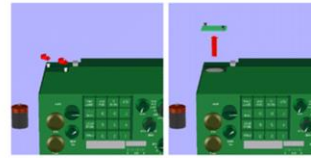
This mismatch can in part be explained by the general utility that most modeling environments strive to achieve. Cinematic terminology is largely derived from character oriented shot compositions, such as over-the-shoulder shots, close shots and mid shots. Operating in these terms would require the semantic (rather than just geometric) representation of objects. Furthermore, the problem of translating most cinematographic notions into controllers is non-trivial, for example, even the seemingly simple notion of a shot will encompass a large set of possible, and often distinct, solutions. However, providing users with high-level tools based on cinematic constructs for the specification of cameras and camera paths, would represent a significant advance over the existing key-frame and velocity graph-based controls.

Application: Visualisation - Multimodal Systems

- support user understanding of presented data or procedures
 - or any task the user is performing
- coordination of language and graphics



Remove the old holding battery. Step 1 of 2



Step 1: Remove the holding battery cover plate, highlighted in the right picture. Loosen the captive screws and pull the holding battery cover plate off the radio.



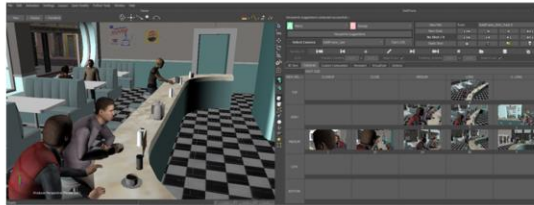
In practice, even partially automated three-dimensional multimedia generation requires an interpretation and synthesis framework by which both the visuospatial properties of a viewpoint can be computed (i.e. the interpretive framework) and the viewpoint controlled according to the constraints arising from the semantics of the language used (i.e. the synthesis framework). Likewise, future scientific and information visualization systems will benefit greatly from intelligent camera control algorithms that are sensitive to both the underlying characteristics of the domain and the task that the user is engaged in. Such adaptive behavior presupposes the ability to evaluate the perceptual characteristics of a viewpoint on a scene and the capability to modify it in a manner that is beneficial to the user.

Beyond simple object references, the coordination of language and graphics poses a number of interesting problems for camera

control. Indeed, such applications are a rich source of constraints on a camera, as the semantics of some spatial terms can only be interpreted by reference to an appropriate perspective. For example, descriptions involving spatial prepositions (e.g. in front of , left of) and dimensional adjectives (e.g. big, wide) assume a particular vantage point. For projective prepositions the choice of a deictic or intrinsic reference frame, for example, for the interpretation of in front, directly depends on the viewpoint of a hypothetical viewer.

Application: Movies

- CG movies
- digital previz
 - tools to aid the prototyping of camera angles and movements



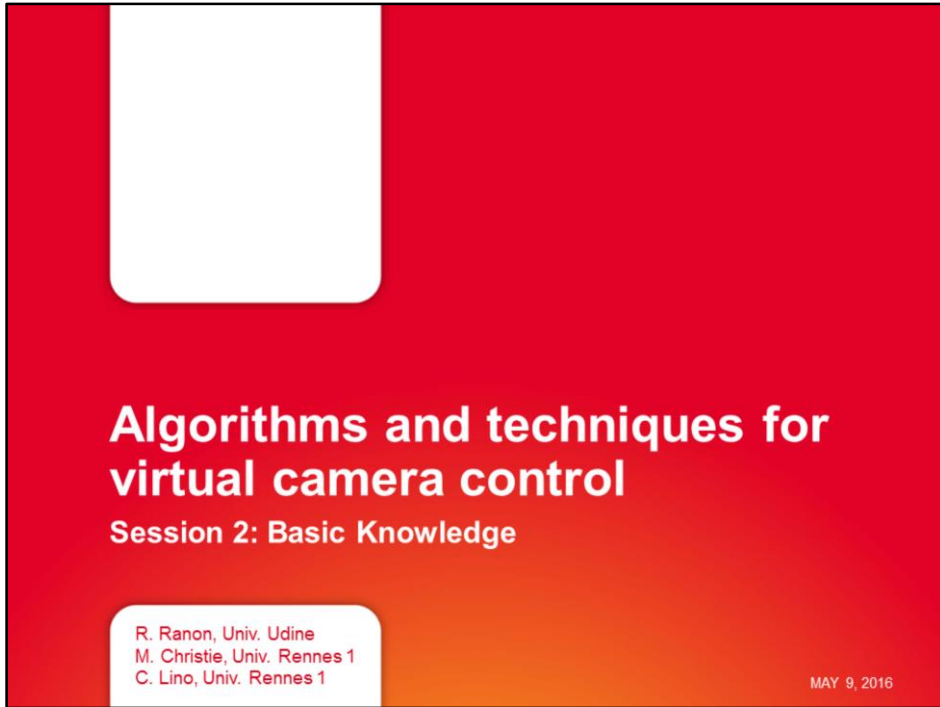
While big-budget CG movies can use professional camera animators and even real-cameras motion capture, low-budget CG movies are becoming more and more practical as the rendering capabilities of games engines (UE4, Unity) progress towards large scenes with realistic global lighting and cinema-level post process effects. In this context, it make also sense to develop camera control algorithms that can aid the user in quickly placing and moving cameras, as well as editing the final result. The same need is even more pressing in the context of previz tools, in which one should be able to quickly preview camerawork into a digital, simplified version of the film set. Sophisticated camera control and editing algorithms are thus key to the realisation of a new generation of storyboarding tools that allow the cinematographer to “prototype” a movie.

Tutorial Software

- Unity Viewpoint Computation Library:
<https://github.com/robertoranon/Unity-ViewpointComputation>
- ToricCam library:
<https://sites.google.com/site/christophelino/libraries/toric-cam>
- “Back To The Future” data set,
<https://cinematography.inria.fr/resources/continuity-editing-for-3d-animation/>



ENJOY!



In this session we will see some foundations of virtual camera control and I will also give some basic definitions for modelling key aspects.

But before talking about virtual cameras, let first see how a real camera is controlled.

How to represent real cameras ?

- Operator / mechanical system
 - Position / orient the camera
- Physical considerations
 - Shape (comprising the operator / system)
 - Volumes and masses
- Intrinsic camera parameters
 - Lens type, sensor size, aperture, ...
- Optical considerations (lens)
 - Brings image distortion
 - And depth of field

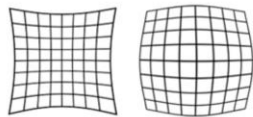
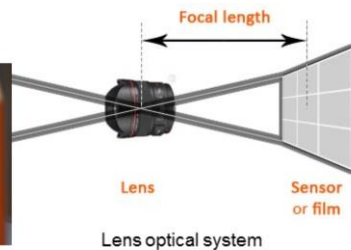


Image distortion



Depth of field effect



Lens
Sensor or film
Lens optical system



When considering a real camera, there is a great number of settings that can be accounted for.

First, the camera is held by an operator or by a mechanical system such as a dolly or a crane.

The camera has a globally non-deformable shape and volume, it has a mass, and while manipulating the camera one should also account for the deformable shape and the mass of the operator or the mechanical system.

In the same way, there are also a number of intrinsic aspects to consider. Indeed, the camera is capturing the world through a physical lens – which one can possibly change between shooting sessions – which will provide a means to project the scene content onto a sensor, which in turn will allow creating a 2D image of the captured scene. The final projection depends on a set of parameters such the sensor size or aperture of the camera (which plays with the amount of light entering the camera). But this projection also come with some optical side-effects, namely an image distortion, due to the shape of the lens, and a blur effect which is called the « depth of field » and is linked to the focal length, i.e. a distance defining how the optical system converges or diverges light.

These multiple complex aspects are currently addressed in as many fields as computer animation, computer vision, or robotics. In this tutorial we will only consider a subset of them, those which are commonly addressed in computer animation.

Model of a virtual camera

➤ « Ideal » camera model (pinhole)

- No operator (free camera)
- No mass, no shape and volume (single point)
- No image distortion (ideal screen projection)

• 7D camera (offered in any 3D modeler)

- 3D world **position** : X, Y, Z
- 3D world **orientation** (pan, tilt, roll)
 - 3 Euler angles : θ, ϕ, ψ
 - Or a quaternion
 - Or a look-at direction ($\psi = 0$)
- 1D zoom
 - Field of view angle : ϕ
- **Aspect Ratio** (fixed)
 - Screen width / Screen height

In the world space

In the camera space

Diagram details: The top diagram shows a camera with axes X, Y, Z and rotation angles θ and ϕ . The bottom diagram shows the view frustum in camera space with origin at the 'EYE or COP (Center of Projection)'. It includes labels for 'UP (y)', 'AT (-z)', 'zNear', 'zFar', 'height', 'width', 'aspect = width/height', and 'fovy'.

Algorithms and techniques for virtual camera control
May 9, 2016

In fact, to simply the problem of handling cameras a bit, in the animation community what we use (and what can basically be found in any 3D modeler) is a pinhole camera model. This is an « ideal » model in that it considers the camera does not have any lens and that the camera is reduced to a single point without a mass. What it means is that the camera will be totally free to move in space and there will be no side-effect coming from the optical system while projecting the scene geometry onto the screen.

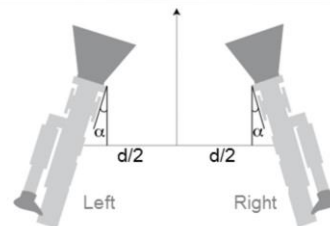
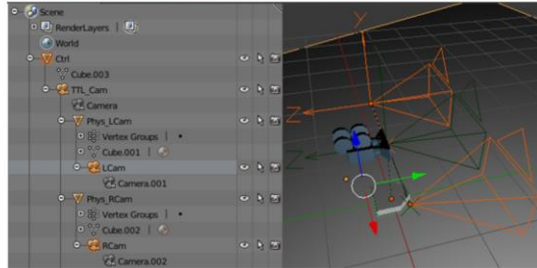
So what we will handle is the 7 essential camera parameters : the 3D camera position, defined in cartesian space; the 3D camera orientation, though it is basically described with three Euler angles (pan for the left-right rotation, tilt for the up-down rotation, and roll for the rotation around the camera axis), due to gimbal lock it is often handled through quaternions in 3d modelers and rendering engines; another way to fix the camera orientation is by providing a look-at direction or look-at point, from wich the camera orientation is computed, also ensuring that the roll angle of the camera is set to zero (assuming that we provided the camera with a proper up vector, i.e. pointing up, fopr instance here the up vector should be Z). The last camera parameter is the zoom factor, which is often handled as a field of view angle (the wider the more we capture scene geometry). And the aspect ratio (i.e. the ratio between the screen width and height) is considered as fixed (common values are

3:2, 4:3 or 16:9).

And these three elements defines the camera projection matrix which is simply computed as the product of three matrices (one for zoom, one for rotation and one for translation).

Possible extensions to our basic camera model

- Simulate a mechanical system
 - Built upon the **scene graph** and **user inputs**
- Example of a 3D stereo camera
 - Build a camera « rig »:
 - 2 cameras (left / right eye)
 - Control : **inter-axis + convergence angle**
 - ⊗ **Consistency has to be enforced by hand !**
 - Depth of field
 - Area of « focus » (interval of distance)
 - **Shader** (blur objects which are out of focus)



Virtual 3D stereoscopic camera rig



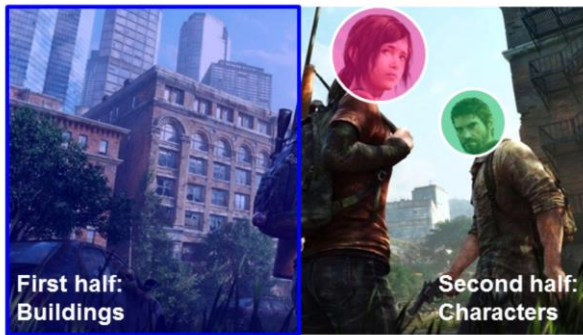
More practically, if we want to build a virtual camera system closer to a real one, or to build a virtual stereo camera it is now possible to rely on the scene graph. For instance we can easily use successive nodes to handle the joints of an articulated arm that reproduces a crane. Or in the same way, we can quite easily build a stereo camera rig by linking two cameras together, which will then add two new parameters to the camera system (namely an inter-axis distance between both cameras center of projection and a convergence angle which however need to be artificially kept consistent).

As for depth of fields, it is also possible to handle this aspect by using a shader that will blur out-of-focus objects, while keeping in-focus objects sharp.

But in this tutorial we will assume we are not handling such parameters, so now let's come back to our simple pinhole camera model, and see how we can manipulate the camera.

Automated viewpoint computation

Specification of the desired shot:



Specified features F^i
2D space
(Screen)



Search
7D Space \mathcal{C}
(Camera configurations
Positions X, Y, Z
Orientations θ, φ, ψ ,
Field of Views ϕ)

Best 7D camera configuration $c \in \mathcal{C}$ satisfying all F^i ?



Algorithms and techniques for virtual camera control

May 9, 2016

The core problem in controlling cameras is the one of visual composition, i.e. deciding what parts of the scene geometry we would like to see and how would like to arrange them onto the screen.

We can for instance want some buildings to appear on the left of the screen, the woman to appear on the top-middle, and the man on the right.

The main difficulty in controlling the camera come from the fact that such a visual composition is given as a set of 2D constraints (in the screen space) and we then need to determine all the 7 parameters of the camera so that the resulting viewpoint can satisfy the desired composition. This make the search problem strongly non-linear.

Interactive viewpoint computation

Input device I

mapping

Camera parameters C

- **Classical Mouse + Keyboard**
 - Dedicated to object inspection and scene exploration
- **Based on Motion Capture**
 - Mimic real camera work
 - Direct mapping to camera parameters
- **Goal: control the shot composition**
 - Direct / indirect control of image features F^I
 - Many interaction metaphors

[Khan et al. 2005]

Optitrack

What practical tools to speed-up user tasks ?

Algorithms and techniques for virtual camera control

May 9, 2016

If we want to move the camera, a first way to do is by directly letting the user handle the camera parameters.

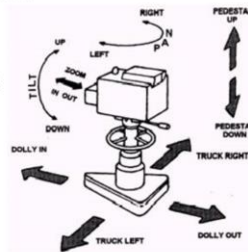
The process relies on the use of an input device offering a number of degrees of freedom to the user, and by handling these degrees of freedom the input values will be mapped (straightforwardly or not) onto output camera parameters.

There have been a large number of such mappings provided in the literature. We can divide them into two main categories: those relying on mouse/keyboard interfaces which have mainly been designed in mind for object inspection or scene exploration tasks, and some more recent mappings relying on post-WIMP interface such as here a virtual camera device based on motion capture that enables the user to handle the camera as a real operator.

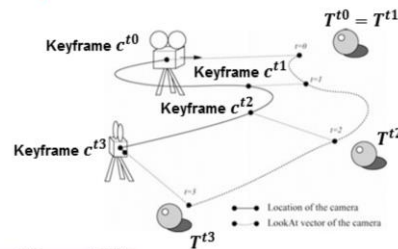
The question raised behind this interactive viewpoint computation is how can we practically help users in their creative and technical tasks?

Computation of camera paths

- Strongly coupled with camera models
- Classical approach (3D animation)
 - Quaternion trajectories (slerp)
 - Linear / spline-based interpolation
- Traditional motions (cinema)
 - Uses tripods / articulated arms
 - Pan/Tilt
 - Pedestal
 - Dolly, Track
 - Crane / Boom
 - ...



Naturally creates smooth camera motions



Another way to move camera is by relying on an automated computation process to create camera paths.

The models for representing camera paths in 3D modelers are still strongly coupled with camera models (i.e. the data representation levels). The user has to define key camera configurations by putting keyframes in the timeline then use an interpolation algorithm. The classical interpolation approach is to rely on quaternions trajectories (Slerp) and linear or spline-based interpolations of the camera parameters (position, orientation and zoom) as it is for animating any 3D object of the scene.

However, as we have seen a bit earlier, real cinemas commonly use tripods or articulated arms to create smooth trajectories such as pan, tilt, pedestal, dolly, track, crane, boom, etc.

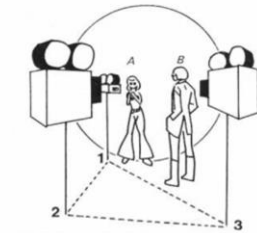
So, a question that raises here is how could we represent and generate such traditional camera motions?

Interactive and automated editing

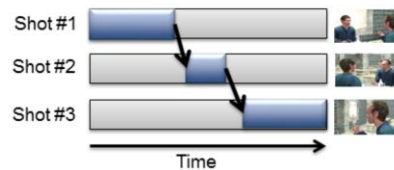
- Shots
- Cuts



- **Where** to cut to ?
 - Best next viewpoint ?
 - Continuity rules, cinema conventions
- **When / Why** to cut ?
 - Best moment to cut ?
 - Pace, occlusions, actions, ...
- Traditional editing (cinema)
 - Rely on « cook-books » / empiric rules
 - Shoot rushes + cut + paste



Shooting and editing dialogues
[Arijon 76]



A final concern in controlling cameras is how to handle the editing. Editing is the process of selecting shots and linking them by introducing cuts.

This first requires to choose a sequence shots which respect some continuity rules and follows cinema conventions. For instance a classical rule is the one related to the line of interest (LoI) which can be represented as an imaginary line drawn between two characters. Crossing this line would change the relative positions of characters on the screen so, in real films, directors rely on a set of cookbooks providing practical rules on how to place cameras around characters and how to make cuts between such cameras.

The editing process then requires to find the best moment to cut (i.e. decide when to cut and above all why to cut to another camera).

This is thus a fairly high level process which raises question about how to model this cinematic knowledge and how to interactively or automatically create good edits.



inria
Informatiques mathématiques

Algorithms and techniques for virtual camera control

Session 3: Interactive Camera Control

M. Christie, Univ. Rennes 1
C. Lino, Univ. Rennes 1
R. Ranon, Univ. Udine

When a camera becomes interactive...

...we need to understand:

- the nature of the mapping between the user inputs and the camera parameters (internal constraints)
- the effect of other constraints on the camera parameters (i.e. external constraints such as visibility or surface of objects)



« Interactive » here is taken in the sense that the user is interacting with the camera (ie manipulating features).

And there are two key questions:

-how is the mapping going to be performed between the user inputs and the camera parameters (which in turn asks the question of which camera model)? This essentially depends on the type task to perform, the nature of the environment, the importance of precision and accuracy, but also aspects such as the cognitive load of the user (and how camera manipulation is critical in performing the task)

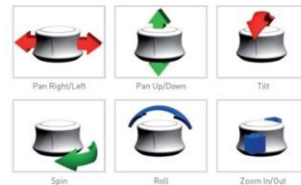
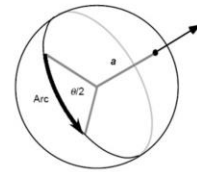
-what is the influence of external constraints on the camera parameters (object geometry, scene complexity, visibility etc)?

And how these constraints guide or counter the user, eg simple collision detection will block the camera (thus the user) from going through a wall but won't prevent him from getting stuck against in front of these walls? So how can the geometry guide the user in his task?

Interactive camera control

4 properties broadly characterize the space of interactive camera control techniques:

- degrees of freedom of the input device
 - low degree of freedom input devices (e.g. virtual arcball [Sho92], [CMS88])
 - 6 degree of freedom input devices (direct metaphors)
- directness of the mappings
 - control camera parameters, velocity, acceleration,...
- nature of the constraint on motion:
 - physical metaphors
 - geometrical
 - task
- world space vs. screen space based control



In the domain of camera control, literature displays a large range of mappings between user inputs and camera parameters. Direct mapping techniques will associate inputs (mouse coordinates) directly to camera parameters, while indirect techniques will operate through specific interaction widgets (e.g. I-widgets [Singh06]) or spaces (screen-space [TTLCC] or application-specific space).

Augmenting usability

How? by reducing the dimensionality of the problem

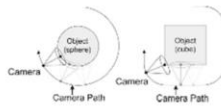
- Fixing camera parameters (e.g. roll parameter)
- Automatically computing camera parameters
 - Lookat of the camera fixed to a target
 - Adding physical constraints to the camera
- Constraining camera parameters to a sub-space of possible motions
- Proposing new camera models



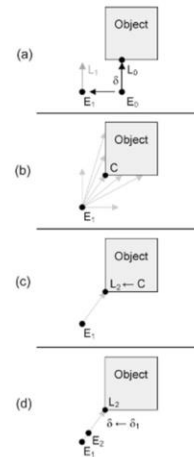
Techniques have rapidly introduced constraints to augment the usability by assisting the computation of some degrees of freedom. This is typically addressed by reducing the dimensionality of the control problem, and/or the application of physics-based models, vector fields or path planning to constrain possible movement and avoid obstacles [HW97]. For example, the application of a physical model to camera motion control has been explored by Turner et al. [TBGT91]. User inputs are treated as forces acting on a weighted mass (the camera) and friction and inertia are incorporated to damp degrees of freedom that are not the user's primary concern.

Constraints in proximal navigation

Khan et al [KKS*05] developed a “hovercam” metaphor for individual **object inspection**:



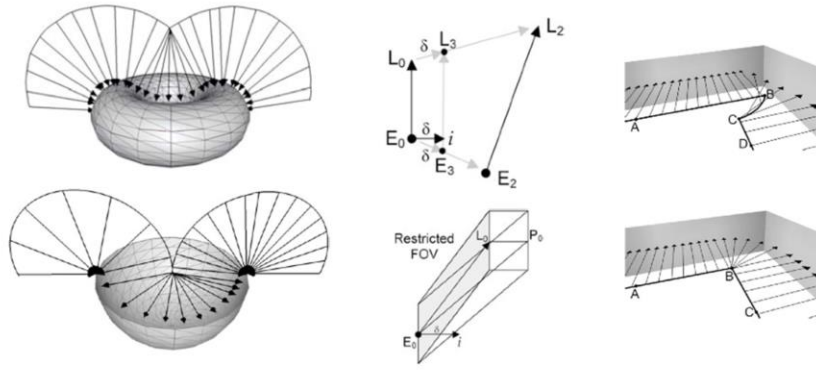
- apply user input to the eye point E_0 (current camera position) and look-at point L_0 , to create E_1 and L_1 ;
- search for the closest point C on the object from the new eye position E_1 ;
- turn the camera to look at C , and
- correct the distance δ_1 to the object to match the original distance to the object δ to generate the final eye position E_2
- clip the distance travelled



This slide illustrates interactive approaches related to object (referred as proximal inspection) and environment exploration. A certain knowledge of the environment is utilized to assist the user in his navigation or exploration task. Such approaches are split according to their local or global awareness of the 3D scene.

Khan et al. [KKS+05] propose an interaction technique for proximal object inspection that automatically avoids collisions with scene objects and local environments. The hovercam tries to maintain the camera at both a fixed distance around the object and (relatively) normal to the surface, following a hovercraft metaphor. Thus the camera easily turns around corners and pans along at surfaces, while avoiding both collisions and occlusions. Specific techniques are devised to manage cavities and sharp turns.

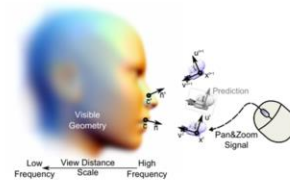
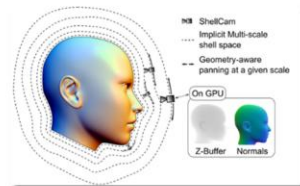
Constraints: Khan [KKS*05]



Left, top and bottom: negotiating bumps and holes in proximal inspection
Right, top and bottom: negotiating corners

Constraints: Shellcam [Bbk14]

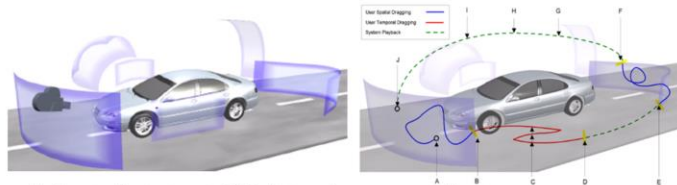
- Boubakeur extended the approach using a smooth motion subspace on arbitrary objects
- A scale-dependent offset shell is computed around the geometry
 - it provides tangent directions for pan/tilt camera motions
 - the zoom changes the offset shell
- The *shell* is a low frequency offset of the geometry



This work can really be viewed as a generalization of the Hovercam, and removes a number of tweaks and limitations the technique had. The idea consists in computing offset shells around the geometry and having the camera navigate on these shells, or traverse them. The distance to the geometry defines the frequency of the offset shell (close=high frequency, so follows closely the details on the surface, far= low frequency so follows a smoothed representation of the surface). Shells are dynamically computed in the vicinity of the camera (not as a precomputation), making the techniques adaptable in any 3D modeller

Constraints: Burtnyk [BKF*02]

- Burtnyk et al. [BKF*02] developed a system to constrain the camera motions to authored surfaces which are linked with hand-built transitions

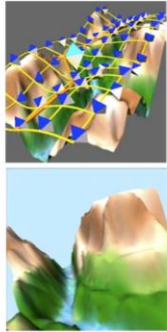


- Interaction as a 2D dragging on surfaces:
 - continuous drag between surfaces and transitions
 - transitions are triggered at specific locations on the constraint surface

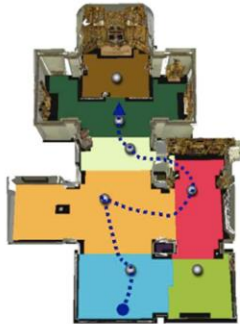


In more stylistic way compared to [KKS*05], Burtnyk et al. [BKF+02] propose an approach in which the camera is constrained to a surface defined around the object to explore (as in [HW97]). The surfaces are designed to constrain the camera to yield interesting viewpoints of the object that will guarantee a certain level of quality in the user's exploratory experience, and automated transitions are constructed between the edges of different surfaces in the scene. The user navigation freely in the bounds of the *constraint surface*, and on reaching an edge is guided to another constraint surface, or hand-built transition.

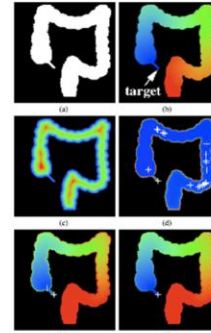
Constraints in interactive exploration



Hanson et al [HW97]



Andujar et al [AVF04]



Hong et al [HMK*97]



We now detail techniques that rely on the geometry of the whole environment to build constraints, that assist the users in either navigation or exploration tasks.

Environment-based control

- methods to assist navigation/exploration are mostly based on motion planning techniques from the field of robotics:
 - potential fields and vector fields
 - cell-and-portal decomposition
 - regular or probabilistic roadmaps
- methods require heavy pre-computations
 - in cost (cell-and-portal) or in storage of data structures (probabilistic roadmaps)
- (generally) inappropriate for dynamic environments



Environment-based assistance, for which applications are generally dedicated to the exploration of complex environments, requires specific approaches that are related to the more general problem of path-planning. Applications can be found both in navigation (searching for a precise target) and in exploration (gathering knowledge in the scene). Motion planning problems in computer graphics have mostly been inspired by robotics utilizing techniques such as potential fields, cell decomposition and roadmaps.

Potential-field constraints

Principle:

- originated in theoretical physics from the study of charged particle interactions
- in the 3D world:
 - obstacles are considered as repulsive potentials
 - targets are considered as attractive potentials
 - the field function F is defined as the aggregation of all potentials
 - the solving process is based on a set of local moves following the steepest descent. The gradient of F gives the direction of the move:

$$M(p) = -\nabla F(p)$$

- potential field can be locally updated [Bec02]



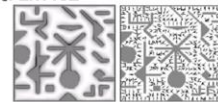
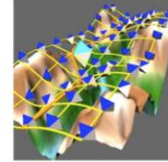
The low cost of implementation and evaluation of potential fields make them a candidate for applications in real-time contexts.

The efficiency of the method is however overshadowed by its limitations with respect to the management of local minima as well as difficulties incorporating highly dynamic environments. Nonetheless, some authors have proposed extensions such as Beckhaus [Bec02] who relies on dynamic potential fields to manage changing environments by discretizing the search space using a uniform rectangular grid and therefore only locally re-computing the potentials.

Vector-field constraints (1)

Extension of potential fields [HW97, XH98]:

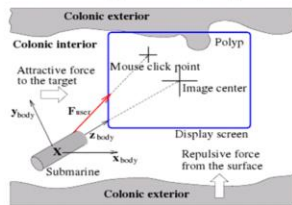
- requires a **constraint surface**: navigation surface that establishes the reachable areas in the environment
- requires a **camera model field**: to each point of the constraint surface is attached some orientation constraint
- a bi-cubic spline interpolation is used to interpolate between the user specified orientation constraints
- possibility to automate the process [TC01]



In [HW97], the constraint surface is defined by the user, together with a number of orientation key-points. Recent approaches consider automated computation of either scalar or vector fields to assist the users both in location and orientation [TC01, ETT07]. This requires to answer a number of key issues (handling bottlenecks such as narrow doorways, handling large open spaces, identifying essential landmarks that make this problem a difficult one).

Vector-field constraints (2)

- Application to virtual colonoscopy [HMK97]:
 - repulsive forces to maintain the camera away from the colonic surfaces: $V(\mathbf{X})$
 - attractive forces towards the target: $V(\mathbf{X})$
 - energy dissipation factor: k_f
 - rigid body dynamics to handle user interaction: \mathbf{F}_{user}



$$\dot{\mathbf{P}}(t) = -\nabla V(\mathbf{X}) - k_f \mathbf{P}(t) + \mathbf{F}_{user}(t),$$



Virtual endoscopy enables the exploration of the internal structures of a patient's anatomy. Difficulties arise in the interactive control of the camera within the complex internal structures. Ideally important anatomical features should be emphasized and significant occlusions and confined spaces avoided. The underlying techniques mostly rely on skeletonization of the structures and on path planning approaches such as potential fields. For example, [HMK97] and [CHL+98] report a technique that avoids collisions for guided navigation in the human colon. The surfaces of the colon and the center line of the colon are modeled with repulsive and attractive fields respectively.

In [HMK97], the camera is guided by some repulsive forces from the colonic surface, attractive ones that push the camera towards a given target, and user inputs (when pointing an area on the surface). The process is however very specific to the problem (a more general geometry would lead to many cases of failure or inappropriate guidance).

Towards indirect interaction

- multiple approaches implement more elaborate interactions with the camera (*i.e.* from parameters manipulation to properties manipulation)
 - through-the-lens techniques: interaction is performed on the content of the screen (for specifying camera motions, or screen composition)
 - reactive techniques: control is operated over targets which indirectly control the camera motions (typically following avatars [LC08,HHS01])

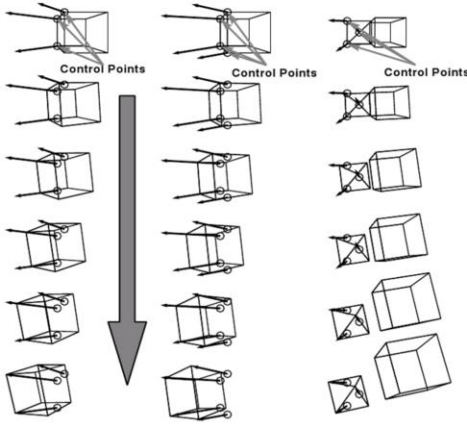


In moving further away from the direct manipulation of camera parameters, through-the-lens techniques enable the control of the screen content

“Through the lens” control

- indicate desired positions of objects on the screen: **through the lens camera control** (Gleicher & Witkin [GW92])
- difference between the actual screen locations and the desired locations indicated by the user is treated as a velocity
- relationship between the velocity ($\dot{\mathbf{h}}$) of m displaced points on the screen and velocity ($\dot{\mathbf{q}}$) of camera parameters expressed with Jacobian that represents the perspective transformation:
$$\dot{\mathbf{h}} = J\dot{\mathbf{q}}$$
- improved complexity of the solving due to Kung et al [KKH95]

“Through the lens” control



“Through the lens” control

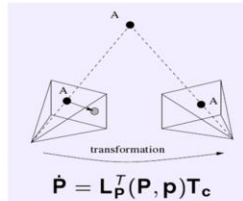
- multiple primitives (lines, circles, paths) can be used to control the composition [YCL08]



- a mass-spring interaction metaphor dampers the users manipulations (often over-constrained)
- composition can be further manipulated by interacting with light sources, shadows and penumbras together with camera parameters

Visual servoing techniques

- use robotics techniques to control camera motions through screen-space constraints [MC02]
- the relation between the changes in 3D environment and the changes in the 2D projected image is expressed in the image Jacobian



- key idea is to invert the equation (constraining the velocity of the key features, compute the camera velocity)



Visual servoing techniques relies on the regulation in the final image of a set of visual features (points, segments, lines).

The image Jacobian (L) expresses the link between the motion of a visual features (P) in the 2D screen and the motion of the camera (it's a linearization of the projection relation for the camera configuration).

The key idea is then to invert the equation, in order to express the variation on camera parameters that correspond to a desired motion of the visual feature on the screen. For exemple, in order to constrain a mobile 3D point at a given location on screen, requires to solve $Jq=0$ at every frame.

Visual servoing techniques

- invert of the Jacobian? generally non-square
 - compute its pseudo inverse with a Singular Value Decomposition
- secondary tasks can be handled given that the primary task doesn't use all *dofs*
 - path coherency
 - collision detection
 - constrain to cinematographic motions (travelling, ...)
- solving process:
 - computation of the Jacobian + SVD decomposition
 - minimization of secondary targets (gradient-based technique)
- visibility needs to be handled separately
 - by excluding some areas from the camera *dofs*



The Jacobian matrix is generally non square ($m \times n$):

- m is the number *dofs* of the camera (7 for euler-based, 8 for quaternion-based)

- n is the number of parameters of the visual features in 2D (2 for a point, 3 for a line, 4 for a segment)

The pseudo inverse of the matrix can be computed by Singular Value Decomposition which is in $O(mn^2)$.

If all camera *dofs* are not constrained, one can perform secondary tasks (see details in next slide) through a minimization process.

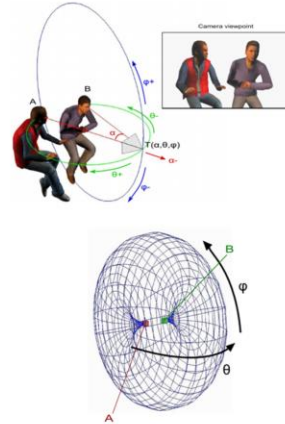
Solving process is quite efficient (cost of Jacobian + SVD + minimization).

However:

- difficult to balance between primary and secondary tasks
- some tasks cannot be easily expressed as a minimization process (visibility/occlusion)

Though the lens control with The Toric Space

- Introducing a novel 3DOF representation of a camera [LC15]
 - dedicated to viewpoint manipulation of two targets
 - Three parameters to control the position:
 - α : angle between targets A and B
 - θ : horizontal angle
 - φ : vertical angle
 - the framing of the two targets is implicitly defined in the model
- (Unity and C++ code available: ToricCam)*



Toric space is a novel representation for manipulating two targets in a screen (and for other camera control tasks as we'll see later).

The idea behind the toric space is a generalized model (in that the model encompasses constraints). These constraints are the on-screen locations of two targets.

3 angles are then defined in this space: alpha, representing the angle between the targets and the camera, theta, the horizontal angle, and phi the vertical angle.

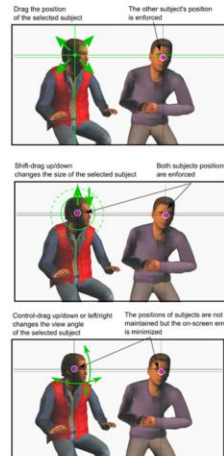
By changing values of phi and theta, the camera moves, but the constraint remain satisfied (ie whatever value of phi, theta) the targets project at the same location on screen.

Code is available here: <https://sourceforge.net/projects/toric-cam/>

Manipulations in the Toric Space

Principle:

- **Manipulation of one target:**
 - while the other is constrained in the screen-space
 - and roll is constrained to 0 (or a fixed value)
- **Interactions:**
 - change on-screen positions, distances, and vantage angles
 - example for on-screen positions:
 - we search for a position on the manifold surface where roll is null and minimizes the change in on-screen position



$$\min_{(\theta, \varphi)} (p_A - p'_A)^2 + (p_B - p'_B)^2$$



Bibliography

- [XH98] D. Xiao and R. Hubbard, *Navigation Guided by Artificial Force Fields*, Proceedings of ACM CHI 98, 1998
- [YCL08] Y. Tao, M. Christie, and X. Li, *Through-the-lens Scene Design*, Proc. of 8th International Conference on Smartgraphics, 2008
- [MC02] E. Marchand and N. Courty, *Controlling a camera in a virtual environment*. The Visual Computer Journal, 18(1):1-19, February 2002
- [Sho92] Shoemake K.: *Arcball: a user interface for specifying three-dimensional orientation using a mouse*. In Proc. of Graphics Interface, 1992.
- [KKS 05] Khan A., Komalo B., Stam J., Fitzmaurice G., Kurtenback G.: *Hovercam: interactive 3d navigation for proximal object inspection*. In Proc. of the 2005 symposium on Interactive 3D graphics, 2005.
- [BKF* 02] Burtnyk N., Khan A., Fitzmaurice G., Balakrishnan R., Kurtenbach G.: *Stylecam: interactive stylized 3d navigation using integrated spatial & temporal controls*. In Proc. of the ACM symposium on User interface software and technology, 2002.
- [HW97] Hanson A., Wernert E.: *Constrained 3d navigation with 2d controllers*. In Proc. of the IEEE Visualization Conference, 1997.
- [HMK* 97] Hong L., Muraki S., Kaufman A., Bartz D., He T.: *Virtual voyage: interactive navigation in the human colon*. In Proceedings of the Siggraph Conference, 1997.
- [Bbk15] Boubekur, T. ShellCam: Interactive Geometry-Aware Virtual Camera Control IEEE International Conference on Image Processing 2014
- [LC15] Lino C. and Christie, M. "Intuitive and Efficient Camera Control with the Toric Space" Proceedings of SIGGRAPH 2015.



inria
Informatiques mathématiques


Algorithms and techniques for virtual camera control

Session 4: Viewpoint Computation

M. Christie, Univ. Rennes 1
C. Lino, Univ. Rennes 1
R. Ranon, Univ. Udine

Viewpoint Computation

- given:
 - a camera model (e.g., position - orientation - FOV), and a domain $D \subset \mathbb{R}^7$ of allowed camera parameters
 - requirements about the visual composition of targets in the computed image
- compute a value for each camera parameter to (best) satisfy the requirements



This is the classical form of the viewpoint computation problem as reported in several papers, e.g. [Olivier et al. 1999, Bares et al. 2000, Christie and Normand 2005, Burelli et al 2008, Ranon and Urli 2014]. In some cases, the problem could be reduced in its dimensions, e.g. because some degrees of freedom, or the FOV of the camera are fixed in advance.

Example



requirements: houses 1 and 2 completely visible, seen from the front; houses area on screen each about 10%



This is an example solution to a viewpoint computation problem, where requirements about visibility, and angle between camera and houses, are fully satisfied: there are no objects between the camera and the houses, and we can see both houses from the front. However, since the houses are at quite different distances from the camera, it is impossible to fully satisfy both the angle and projected area requirements. Another solution could have instead framed the houses from a different angle, and try to instead make them have the same projected area.

Approaches to VC

- **algebraic**: when we can establish an algebraic relation between requirements and camera parameters
 - works only in very limited situations
- in all other cases, we can use:
 - **constraint-based** approaches: express requirements as constraints over D , find camera parameters \mathbf{c} that satisfy constraints, or fail
 - **optimisation-based** approaches: express requirements as a satisfaction function $F:D \rightarrow [0, 1]$, find camera parameters \mathbf{c} that maximise F



Algebraic approaches (e.g. [Blinn, 1988]) work only for 1-2 targets and are not able to take into account some kind of requirements, most notably visibility, since it is a property which depends on the spatial layout of the whole scene. As such, they are of very limited use.

Constrain-based and optimisation approaches do not exhibit such limitations and generally can work with an arbitrary number of targets and any kind of properties that can be expressed through constraints or satisfaction functions. We focus, in the following, on optimisation approaches since they have the nice ability to compute a solution even when the problem is over-constrained, i.e. when the visual properties cannot be all satisfied. This

situation is far more common than one may think, since in a dynamic environment, targets can easily be in configurations that make a VC problem not perfectly solvable.

Visual Composition Requirements

- we consider the following types of visual composition requirements:
 - **size** (width, height, area) of targets on screen
 - **visibility** of targets on screen
 - **angle** of targets with camera
- we need to:
 - model each type of requirement as a satisfaction function $f:D\rightarrow[0,1]$
 - model the satisfaction function of a virtual camera \mathbf{c} as some composition of the requirement satisfaction functions, i.e. $F(f_1, f_2, \dots, f_n):D\rightarrow[0,1]$



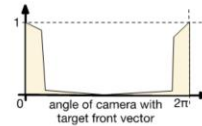
We consider a basic set of visual composition requirements, however sufficient to express a wide range of application needs, to explain the process of turning them into satisfaction functions. Other types of requirements can be quite easily modelled, and can also include aesthetics features such as balance, rule of the thirds, and so on. For example, rule of the thirds has been used in [Abdullah et al 2011, Bares 2006], and balance has been modelled in [Abdullah et al 2011]. It is also possible, as shown e.g. in [Olivier et al. 1999] to model requirements that involve two or more targets, e.g. “target T1 should be seen to the right of target T2” or “target T1 should be smaller than target T2”.

Modeling requirement functions

- a requirement has a type (size, visibility,...) and a desired value

- the “type” part computes the value v of a visual feature (size, visibility, angle) of a target t for a given camera, i.e. $f_{\text{type}}(\mathbf{c}):D \rightarrow V$ where V is the set of possible values of the visual feature

- the “desired value” part computes a satisfaction value from the value of the visual feature, i.e. $f_{\text{desired}}:V \rightarrow [0, 1]$ and can be e.g. modelled as a linear spline



example f_{desired}

$$F(\mathbf{c}) = f_{\text{desired}}(f_{\text{type}}(\mathbf{c}))$$

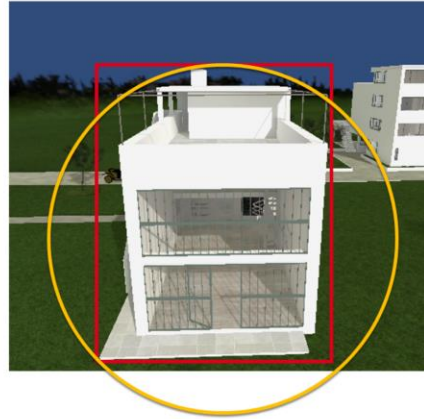
- optimisation approaches typically need to sample a considerable number of points in D to find a good solution: therefore, computing f_{type} must keep into account accuracy vs cost
- compromise might depend on specific application demands



In some papers, instead of a linear spline, a gaussian function is used, with the goal of smoothing the function around the desired value. In general, this is advisable, since we don't need extreme precision with visual features: for example, it is very hard to distinguish from a projected target area of 0.95 the area of the screen, and a projected area of 0.97 the area of the screen.

Measuring Size (area)

- mesh vs bounding volume
 - what about objects with holes
- rendering (and counting pixels)
- geometrical methods
 - bounding sphere
 - bounding box



rendering at 1000x750	rendering at 500x375	rendering at 80x60	rendering at 40x30	geometric evaluation
261.87	63.47	9.7	8.44	0.005



mean evaluation cost (milliseconds) in a scene after 1000 evaluations from random cameras

To measure area, it is common to use some kind of bounding volume (bounding sphere, AABB, ...), which makes it much easier to perform geometrical calculations, and also works nicely with objects with holes (e.g. a grilled fence), where typically the perceived area on the frame is intended to include those holes. The typical considerations about bounding volume fitting apply (e.g. spheres are better for nearly spherical objects, ...).

There are basically two alternatives to measure size: one is to render the target with a unique color, perhaps at low resolution, and then count the pixels after having moved the rendered image to main memory; the other one is to

use some geometrical computation with the bounding volume. For example, [Ranon and Urli 2014] compute the area of a target t by taking the (oriented or axis-aligned) bounding box of t , finding the vertices of it that are visible from v , and projecting them, using the fast look-up table approach proposed in [Schmalstieg and Tobler, 1999]. The resulting 2D hull polygon is then clipped by the viewport through a standard Cohen-Sutherland algorithm, and finally, as the resulting polygon is convex, a contour integral approach can be used to quickly compute its area.

The table reports average times in milliseconds needed to compute the size of a target in a scene, using rendering at various resolutions and the geometrical approach outlined above. As we can see, cost using rendering, even at very low resolutions, is orders of magnitude greater than a geometrical method, even considering that by using rendering methods, we can measure the size of all targets, instead of just one. The major cost of rendering methods is the transfer of the image to main memory. All technical details about the data reported in the table can be found in [Ranon and Urli, 2014].

For height and width, similar considerations apply.

Measuring Visibility

- mesh vs bounding volume
 - objects with holes
- rendering (and counting pixels)
- geometrical methods
 - ray casting



rendering at 1000x750	rendering at 500x375	rendering at 80x60	rendering at 40x30	geometric evaluation
261.68	63.1	9.61	8.96	0.1



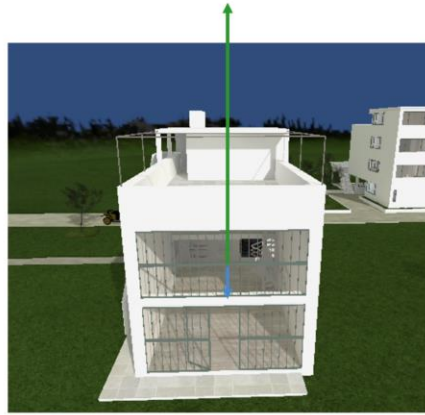
mean evaluation cost (milliseconds) in a scene after 1000 evaluations from random cameras

To measure visibility of a target, there are basically two alternatives: one is to render the scene using a unique color for the target, another color for the scene, and turning on blending, and then count blended and coloured pixels after having moved the rendered image to main memory; the other one is to perform a number of ray casts, e.g. to selected points in the bounding volume of the target or to random mesh vertices. For example, [Ranon and Urli 2014] use 9 ray casts, to the center and corners of the bounding box of the target, and report visibility as the ratio of ray casts which do not cross other objects before reaching the target. Even 6 ray casts are sufficient in most situations.

The table reports average times in milliseconds needed to compute the visibility of a target in a scene, using rendering at various resolutions and ray casting with 9 rays. As we can see, cost using rendering, even at very low resolutions, is at least one order of magnitude greater than the ray casting method.

Angle

- the angle between a target-specific vector \mathbf{u} and the vector from t to the camera
 - typically, \mathbf{u} can be the *forward* vector of the target (horizontal angle between camera and target), or its *up* vector (vertical angle between camera and target)
 - other choices are possible, e.g., for a character, \mathbf{u} could be the direction of the head



An angle requirement, being just the computation of an angle between two vectors, is very cheap to compute.

Computing Satisfaction

- typical solution is a weighted sum of individual f

$$F(c) = \sum_i f_i(c)$$

- corresponds to logical AND of all requirements
- weights allow one to set requirements importance, but are not easy to manage
- can use also other logical operators (e.g. OR is max)



A weighted sum allows one to express the logical AND of all requirements, and weights allow one to control a requirement importance with respect to the others. However, this might not be expressive enough for some situations. Suppose, for example, that the satisfaction of a visibility requirement should be set to zero if the target is off screen, in order to penalise solutions where targets are off screen (recall that the ray casting method for measuring visibility does not check if the target is on screen or not). This cannot be expressed using weighted sums. A recent proposal by Lino [Lino 2015] introduces more sophisticated operators to build F from individual requirements functions, e.g. to cover situations like the one presented above.

Solving VC

- due to complexity of the objective function and non-continuity (e.g., think visibility), black-box optimisation approaches are preferable
- use of random values (stochastic optimisation) to escape local minima
- population-based approaches have the additional advantage that poor initialisation can be corrected
- Particle-Swarm Optimisation (PSO) has all these features and, moreover, it is known for fast convergence
 - used by several authors, e.g. [Burelli et al. 2008, Abdullah et al. 2011, Ranon and Urli 2014]



Black-box optimization approaches are suitable when we can compute the objective function, but we have no analytical expression for it that can be used, e.g., to compute gradients. Combining this with the fact that our search space is quite large, there is the need to adopt stochastic techniques to promote exploration and escaping local minima.

Population-based techniques add to this the usage of several candidates to explore search space. There are a lot of population-based optimization approaches that can be used in VC. Some authors have, for example, used genetic algorithms, e.g. [Olivier et al. 1999]

PSO for VC

- idea: a swarm of cameras wanders through D in search of the optimum (i.e. the parameters \mathbf{c} that maximise F)
- at each step, we move a camera and evaluate F on it
- we always record:
 - the best visited position in D for each camera (\mathbf{p})
 - the global best visited position \mathbf{p}_g
- the equations for moving a camera from its position in D \mathbf{x}^{n-1} to a new position \mathbf{x}^n are:

$$\begin{aligned}\mathbf{v}_i^n &= w^{n-1} \mathbf{v}_i^{n-1} + c_1 r_1 (\mathbf{p}_i^{n-1} - \mathbf{x}_i^{n-1}) + c_2 r_2 (\mathbf{p}_g^{n-1} - \mathbf{x}_i^{n-1}) \\ \mathbf{x}_i^n &= \mathbf{x}_i^{n-1} + \mathbf{v}_i^n \quad i = 1, 2, \dots, N\end{aligned}$$

w , c_1 and c_2 are PSO-specific parameters; r_1 and r_2 are random numbers in $[0,1]$



The idea is that a camera will move both towards the leader camera (the one that found the best parameters so far) and towards its local best found parameters. r_1 and r_2 throw in a bit of randomisation, while c_1 and c_2 are parameters in $[0,1]$ that can be used to balance the importance of the local optimum versus the global one. w is an inertia weight which it establishes the influence of the search history on the current move. A common strategy is to use a decreasing inertia value, from a starting w_{init} to an ending w_{end} value.

In this version of PSO, a single global leader is used; there are variants of PSO that use more leaders, e.g. according to distance from cameras. It is also worth

noting that there are dozens of PSO variants that slightly change the equations by e.g. reducing the number of parameters.

PSO for VC

```
initialize n random cameras in array CAMERAS
i=0;
while (there is still time left) {
    move CAMERAS[i];
    evaluate F(CAMERAS[i]);
    compute new local and global optima;
    i = (i+1) mod n;
}
optimum = CAMERAS[g];
```



At the beginning, we can set each camera local optimum to the initial position in D , and any index as the global optimum g .

The equations in the previous slide do NOT prevent a camera from exiting D . In the case it happens, we can simply set its satisfaction to zero, and it will return in D in successive steps; another option is to clamp the parameters to be inside D .

DEMO



Various demos using our Unity Viewpoint Computation Library, available at <https://github.com/robertoranon/Unity-ViewpointComputation>

Improving PSO

- unlucky random initialisation coupled with little available time (e.g. few milliseconds) and/or large search space can make PSO fail
- current methods to tackle this issue are:
 - “smart” initialisation
 - lazy F evaluation
 - PSO parameters tuning



Even with the mentioned methods, there is no guarantee that a PSO run will find a good camera, i.e., from time to time, bad runs can happen. In such cases, a simple remedy is to restart the PSO.

Smart Initialisation

- size and angle requirements are very common in VC problems
- it is quite easy to initialise a camera such that it roughly satisfies a size or angle requirement (or both)
- e.g., for size, we can compute a roughly optimal distance to a target by the formula

$$\text{distance} = \frac{\text{target size}}{\text{target's projection size}} \cdot \frac{1}{\tan(\gamma/2)}$$

where target size and projection size are easily computed by using a bounding sphere, assumed centered on the screen

- if a problem has k targets, we can distribute cameras among them, and initialise each camera around optimal values for the assigned target



The detailed description of the approach is in [Ranon and Uri, 2014]. Smart initialisation can be mixed with purely random initialisation to improve swarm diversity and thus coverage of D .

Lazy F evaluation

- the evaluation of F can be terminated as soon as we know that we cannot improve on the camera local best value (*lazy evaluation*)
 - the computed value would have no effect on camera movement
- we can then order the requirements by cost of evaluation (angle, size, visibility) so that we avoid computing unnecessary (and costly) requirements
- other strategies are possible, e.g. combine lazy evaluation with computing first the projection of bounding box of all targets, and then set the satisfaction of any requirement for the same target, if the projection is off-screen, to zero



The last method is explained in detail in [Ranon and Urli, 2014].

PSO parameters tuning

- the values of n (the number of cameras in the swarm), c_1 , c_2 , and w can greatly influence the behaviour of PSO
- given a set of scenes and VC problems, and a set of possible PSO parameter values, one can run all possible combinations, and then use statistical methods to derive optimal PSO parameter values
- in our experience, derived parameters are quite good for all similar settings



The influence of PSO parameters tuning is generally underestimated. [Ranon and Urli 2004] proves that, for VC problems, it can make a significant difference. In the following, we review the main steps of their parameter tuning process, which is based on the Friedman rank sum test, and Friedman post-hoc analysis

PSO parameters tuning



Scene	Triangles	Objects	Scene AABB
<i>city</i>	474083	324	300 x 100 x 300 (vol: 9×10^6)
<i>house</i>	324182	50	120 x 23 x 100 (vol: 276×10^3)
<i>rooms</i>	110474	240	13.9 x 3.0 x 21.8 (vol: 909.06)

5 problems per scene

Name	Meaning	Values
N	number of particles in PSO	20, 30, 40, 60, 80, 100, 130, 160, 200, 240, 290, 340, 380]
r_{part}	fraction of randomly initialized particles	[0.0, 0.33, 0.66, 1.0]
c_1	PSO cognitive parameter	[0.0, 0.5, 1.0, 1.5, 2.0, 2.5]
c_2	PSO social parameter	[0.5, 1.0, 1.5, 2.0, 2.5]
w_{init}	PSO initial inertia weight	[0.5, 1.0, 1.5, 2.0]
w_{end}	PSO final inertia weight	[0.0, 0.5, 1.0]

18720 combinations

20 runs per scene, problem, combination = 5'148'000 runs

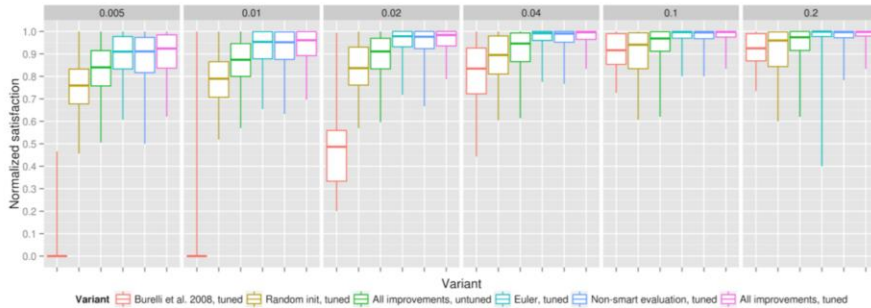
considering 6 time budgets for PSO: 5, 10, 20, 40, 100, 200 milliseconds



The parameters tuning considers three scenes (exterior, interior, mixed) with 5 problems for each scene, ranging from 1 to 5 targets. We consider a choice of PSO parameters from the literature, and, for each choice of parameters, and problem, we perform 20 runs of PSO. We repeat the procedure for 6 different time budgets. Our analysis then ranks the parameter combinations from the best to the worst, and prunes the ones that are statistically inferior.

PSO parameters tuning

T (ms)	How many	Best ($N, r_{part}, c_1, c_2, \omega_{init}, \omega_{end}$)	Restriction on parameters values	Median evaluations	
				full	partial
5	102	20, 0, 2, 1.5, 1.5, 0	$N \leq 30, r_{part} = 0, c_2 \geq 1, \omega_{end} \leq 0.5$	60	13
10	167	30, 0, 2.5, 1.5, 0.5, 0	$N \leq 40, r_{part} \leq 0.33, c_2 \geq 1, \omega_{end} \leq 0.5$	108	43
20	117	30, 0, 2, 2, 0.5, 0	$N \leq 60, r_{part} \leq 0.33, c_2 \geq 1, \omega_{init} \leq 1.5, \omega_{end} \leq 0.5$	176	135
40	144	30, 0, 1.5, 2, 0.5, 0.5	$N \leq 60, r_{part} \leq 0.66, c_2 \geq 1, \omega_{init} \leq 1, \omega_{end} \leq 0.5$	373	294
100	173	130, 0, 2, 2, 0.5, 0	$r_{part} \leq 0.66, \omega_{init} \leq 1, \omega_{end} \leq 0.5$	707	625
200	218	200, 0, 1.5, 2.5, 0.5, 0.5	$c_2 \geq 1, \omega_{init} \leq 1, \omega_{end} \leq 0.5$	1204	1206



The results show that parameter tuning has a significant effect (green vs pink box plots). The graph shows also the influence of smart initialisation (yellow vs pink box plots) and lazy evaluation (blue vs pink box plots). Generally, using 20-30 particles is best for time budgets under 50 milliseconds, and smart initialisation is especially effective when the time budget is very low.

Conclusions

- VC cost is comparable to frame rendering; however, it can be spread among a few successive frames
- all techniques presented in this part, and more, are implemented in the C# Unity Library available at <https://github.com/robertoranon/Unity-ViewpointComputation>
 - quite easy to port to other engines (UE4 port is under way)
 - easily implement your own properties, evaluation methods, solver



References

- [Blinn, 1988] BLINN J.: Where am I? what am I looking at? IEEE Computer Graphics and Applications (July 1988), 76–81. 11, 20, 21
- [Olivier et al. 1999] P. Olivier, N. Halper, J. H. Pickering, and P. Luna, "Visual Composition as Optimisation," in Artificial Intelligence and Simulation of Behaviour, 1999, pp. 22–30
- [Bares et al. 2000] W. H. Bares, S. McDermott, C. Boudreaux, and S. Thainimit, "Virtual 3D camera composition from frame constraints," in *Proceedings of the eighth ACM international conference on Multimedia*. ACM Press, 2000, pp. 177–186
- [Christie and Normand 2005] M. Christie and J.-M. Normand, "A semantic space partitioning approach to virtual camera composition," *Computer Graphics Forum*, vol. 24, no. 3, pp. 247–256, 2005
- [Burelli et al 2008] P. Burelli, L. Di Gaspero, A. Ermetici, and R. Ranon, "Virtual Camera Composition with Particle Swarm Optimization," in *SG '08: Proceedings of the 8th International Symposium on Smart Graphics*, vol. Lecture No. no. 5166. Springer-Verlag, 2008, pp. 130–141.
- [Ranon and Urli 2014] Ranon R., Urli T., Improving the Efficiency of Viewpoint Composition, *IEEE Transactions on Visualization and Computer Graphics*, 20(5), May 2014, pp. 795-807.
- [Abdullah et al 2011] Rafid Abdullah, Marc Christie, Guy Schofield, Christophe Lino, Patrick Olivier. Advanced Composition in Virtual Camera Control. *Smart Graphics*, Aug 2011, Bremen, Germany. 6815, pp.13-24, 2011, Lecture Notes in Computer Science.
- [Bares 2006] W. H. Bares, "A Photographic Composition Assistant for Intelligent Virtual 3D Camera Systems," in *SG '06: Proceedings of the 6th International Symposium on Smart Graphics*, ser. Lecture Notes in Computer Science, vol. 4073. Springer-Verlag, 2006, pp. 172–183
- [Schmalstieg and Tobler, 1999] D. Schmalstieg and R. F. Tobler, "Real-time bounding box area computation," *Tech. Rep. TR-186-2-99-05*, Jan. 1999.
- [Lino, 2015] Christophe Lino. Toward More Effective Viewpoint Computation Tools. *Eurographics Workshop on Intelligent Cinematography and Editing*, May 2015





inria
Informatiques mathématiques

Algorithms and techniques for virtual camera control

Session 5: Camera path planning

M. Christie, Univ. Rennes 1
C. Lino, Univ. Rennes 1
R. Ranon, Univ. Udine

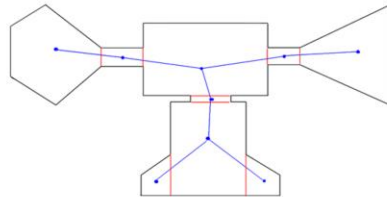
Creating (realistic) camera paths

...is a specific challenge

- it displays the issues the **current path planning techniques** have (how to decompose the environment, how to plan paths)
- and the **issues related to camera control**:
 - ensuring visual on-screen properties along the path (visibility, framing, angle, ...)
 - enforcing smoothness of camera motions/orientations
 - respecting classical features of camera motions

Cell-and-portal decomposition

- performs partitions of the environment into sub-regions (the **cells**), and connections between sub-regions (the **portals**)



- an adjacency graph is built by connecting cells
- camera exploration/navigation tasks can then be casted as a planning process in the adjacency graph [AVF04]

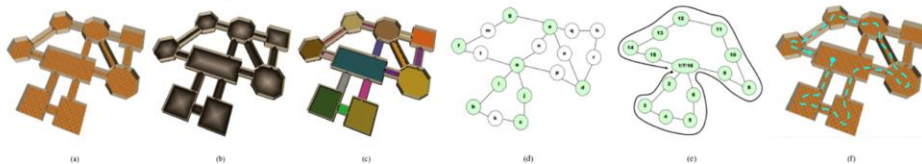


Virtual endoscopy enables the exploration of the internal structures of a patient's anatomy. Difficulties arise in the interactive control of the camera within the complex internal structures. Ideally important anatomical features should be emphasized and significant occlusions and confined spaces avoided. The underlying techniques mostly rely on skeletonization of the structures and on path planning approaches such as potential fields. For example, [HMK97] and [CHL+98] report a technique that avoids collisions for guided navigation in the human colon. The surfaces of the colon and the center line of the colon are modeled with repulsive and attractive fields respectively.

In [HMK97], the camera is guided by some repulsive forces from the colonic surface, attractive ones that push the camera towards a given target, and user inputs (when pointing an area on the surface). The process is however very specific to the problem (a more general geometry would lead to many cases of failure or inappropriate guidance).

Cell-and-portal decomposition

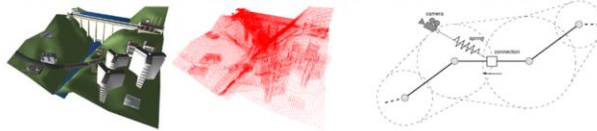
- provides a *structure* to the environment to better perform navigation/walkthrough tasks (the decomposition can be authored)
- Andujar et al. [AVF04] employ this structure to:
 - identify the individual interest of each cell (with an entropy-based metric)
 - compute the sequence of most relevant cells to visit
 - compute a path connecting the cells, portals and relevant viewpoints in the cells



Cell decomposition approaches split the environment into spatial regions (cells) and build a network that connects the regions. Navigation and exploration tasks utilize this cell connectivity while enforcing other properties on the camera. For example, [AVF04] proposed such a technique to ease the navigation process and achieve shots of important entities and locations. Using a cell-and portal decomposition of the scene together with an entropy-based measure of the relevance of each cell, critical way-points for the path could be identified.

Voxel-based decomposition

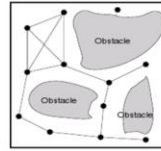
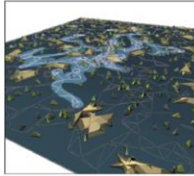
- a regular partitioning of the free space (voxels) can be used to generate guided tours [ETT07]:
 - visibility of (authored) landmarks is computed for each of the voxels in a pre-process
 - all voxels that view at least one landmark are connected together to form an adjacency graph
 - a solving process (Travel Salesman-like) computes the suite of voxels to visit in the graph to ensure that each landmark has been viewed at least once
 - in interactive mode, a memory of the visited landmarks is maintained to guide/constrain the users navigation, through a spring-based physical system



Following an idea similar to Andujar, yet in a more interactive context, Elmquist et al. [ETT07] propose to automate the construction of a navigation graph between user-defined landmarks. The environment is decomposed into voxels, each of which is evaluated for visibility against the landmarks. An adjacency graph is then built between voxels sharing the same landmarks, and explored with a TSP algorithm to compute the best path that visits all the landmarks.

Roadmap constraints

- roadmap planners operate in two phases:
 - first sampling the space of possible configurations
 - second constructing a connectivity graph by linking neighbour samples (and checking for collision on the links)



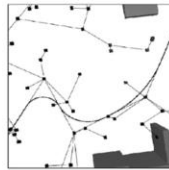
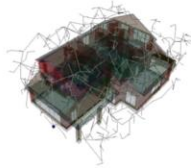
- simple to construct and navigate inside the graph
- complex to determine the appropriate density of sampling (but PRM complexity is a factor of the scene complexity)



Roadmaps, and especially probabilistic roadmaps are a simple-to-implement and efficient technique to perform path planning tasks at the level of an environment. For transition planning (moving from one landmark to another), target tracking and cut-jumping (switching between viewpoints), the process needs to be augmented by visibility computation, either in a static way [NO03], or in a dynamic way [LC08].

Roadmaps in camera planning

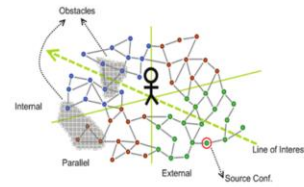
- [NO03] rely on probabilistic roadmap techniques for camera planning:
 - roadmap is consisting of collision-free camera motions (the camera is abstracted as a sphere, the motion as a cylinder)
 - planning is performed with an advanced Dijkstra process (avoids sharps turns)
 - path is smoothed and camera orientation anticipates camera motion



In [NO03] visibility is guaranteed between connected nodes. Such PRMs can be used in an interactive approach by selecting the most appropriate given the current configuration and the user inputs. The main drawback lies in the cost of updating the data structure when considering dynamic elements.

A local/dynamic roadmap

- Using a locally defined probabilistic roadmap [LC08]
 - a probabilistic roadmap is created around the target and moves with the target (camera positions are expressed in polar coordinates)
 - the path planning is performed in the roadmap to move the camera
 - collision/occluded nodes are removed from the graph using a lazy evaluations strategy
 - new nodes are inserted using a density parameter
 - cuts can be performed between regions (by connecting distant edges)



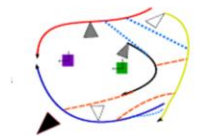
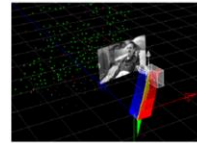
Previous approaches generally suffer from their locality (searching for viewpoints in the local neighborhood of the current camera location). Chang and Li introduce a probabilistic roadmap technique that helps to reduce this locality:

- a roadmap is defined in the local basis of the camera target (the roadmap is built once, and then is only locally modified)
- paths are searched for in this roadmap by evaluating every configuration wrt. visibility of the target and possible collision of the path with the environment:
 - occluded viewpoints and non reachable viewpoints are removed from the roadmap
 - new viewpoints are added when necessary
- in critical situations, cuts can be performed between viewpoints (cuts are represented as expensive edges in the roadmap)

Provides a reactive approach that is more global (lazy-evaluation of the knowledge in connected edges), and allows cuts between paths.

Extract and re-target camera motions

- [SDM14] propose to extract camera targets from movies
 - eg using SIFT-based feature tracking (Voodoo software)
- Trajectories are then re-targeted to the virtual environment (using the ToricSpace)
- All trajectories are then expressed in a motion graph around the targets (similar to [LC08])
 - the graph enables continuous or cut transitions between pieces of trajectories
 - characteristic noise and nature of motions is maintained



This approach is an attempt to retain “realistic” characteristics of “real” camera trajectories and re-use them in virtual environments.

Trajectories are expressed in a “camera motion graph” that is exploited in real-time to determine the best trajectory, and best transitions between trajectories.

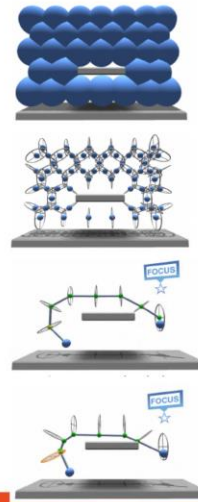
A camera motion graph consists of (i) pieces of original camera trajectories attached to one or multiple targets, (ii) generated continuous transitions between camera trajectories and (iii) transitions representing cuts between camera trajectories. Pieces of original camera trajectories are built by extracting camera motions from real movies using vision-based techniques, or relying on motion capture techniques using a virtual camera system.

A re-targeting is proposed to recompute all the camera trajectories in a normalized representation, making camera paths easily adaptable to new 3D environments.

The camera motion graph is then constructed by sampling all pairs of camera trajectories and evaluating the possibility and quality of continuous or cut transitions.

A global/dynamic roadmap

- Oskam et al [OSTG10] propose a visibility aware roadmap technique:
 - uniform sphere-sampling of the free space in the environment
 - pre-computing sphere-to-sphere visibility (stochastic ray casting)
 - connecting overlapping spheres to build a roadmap
 - planning a rough path from source to target that ensure visibility of a target (focus point)
 - refining the path using rendering-based technique



Authors present a powerful, and simple to implement technique, that can be adapted to many situations. The algorithm first samples the free space with regularly placed overlapping spheres.

Portals (between two spheres) are created where spheres intersect, and a graph is built. Sphere-to-sphere visibility is precomputed (in the static scene) using a stochastic sample process (ie each sphere knows a probability of seeing another sphere. The roadmap is then created, and planning can be performed (eg, classical A*), from an initial given location to a final location, while maximizing visibility of a focus point.

A specific refinement is performed to smooth out the path and to maximize the real visibility of a focus point (the path computed with A* only has estimated visibility)

The roadmap can be locally and dynamically updated when changes in the scene geometry occur

Toric Space interpolations

- Interpolating in the space of visual features
 - introduced by [LC15]
- given to viewpoints v_1 and v_2 :
 - extract visual features (angle between targets, distance to targets, vantage angle of targets) for viewpoint v_1 and v_2
 - perform a linear interpolation of the visual features of the first framing between v_1 and v_2
 - perform a linear interpolation of the visual features of the last framing between v_1 and v_2
 - and then blend between the two trajectories



Interpolating in the space of visual features ... rather than in the space of camera parameters.

While viewpoint interpolation is generally based on spline techniques (one spline for the camera path, and one spline for the camera lookat point is a common representation), in many cases the interpolations fail to maintain visual properties, and animators generally need to tune the spline curves.

In the idea the process is the following:

- a first path is generated between v_1 and v_2 by maintaining the framing at v_1 .
- a second path is generated between v_2 and v_2 by maintaining the framing of v_2
- a blending between both paths is then generated so that at the beginning, the camera maintains the framing at v_1 , and at the end maintains the framing v_2 , and between nicely interpolates the framings.

Visibility: A Fundamental Challenge



- many applications require the visibility of target objects (games, sci. visualization,...)
- importance of visibility (triggers interaction, depth cue, scene understanding, spatial relations...)
- visibility is **application-dependent**
 - a matter of perception (e.g. object recognition)
- visibility has multiple **interpretations**
 - spatial visibility (considering sparse occluders)
 - temporal visibility (with fast moving occluders)



12

Visibility is a central challenge in camera control. Games, for example, require to maintain the visibility of the player and of secondary elements simultaneously (opponents, exits, items,...). Furthermore, games have been operating an important move these last years to a more cinematic experience. In scientific visualization, data may be hidden in a complex geometry setups that evolve over time. In navigation tasks, maintaining the visibility of multiple known landmarks avoids the users from getting lost or losing time in re-orientation.

However visibility is application dependent and has multiple interpretations, which means there is no generic solution to the problem. One can look at the overview proposed by [Elmqvist08] who details techniques to handle occlusion in data and object visualization (however not on how to compute viewpoints that maintain visibility, but on how to alter the geometry or scene graph). This section only considers means to evaluate occlusion and to escape from occlusion.

And Complex Challenge



- two problems:
 - how visible is the target?
 - where should I move the camera to?
- cost of evaluating visibility/predicting motion
 - complexity of the target/complexity of the scene
 - maintenance of visibility data structures
- maintaining visual stability with **sparse or fast-moving** occluders
- integration of visibility computation in the whole camera control process
 - how to balance its influence with other descriptors



The complexity of handling visibility in camera control has many sources:

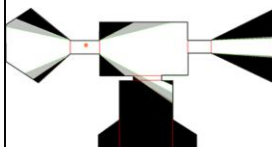
- first of all, the real-time nature of most applications require efficient evaluation AND anticipation of occlusion
- second, maintaining visibility in dynamic environments is computationally expensive (as it is for occlusion culling in the field of visibility techniques for efficient rendering)
- third, the targets are generally complex-shaped objects, for which the estimation of the full visibility is an expensive process.

Handling Visibility

Two classes of techniques for camera control:



- local visibility computation:
 - principle: **sample** or **reason** in a local area
 - with ray-casting techniques
 - with bounding volume intersection
 - with hardware rendering techniques



- global visibility computation:
 - pre-computation from the static geometry (offline)
 - cell-and-portal visibility structure
 - hierarchical cells, ...
 - Followed by an online estimation of visibility



14

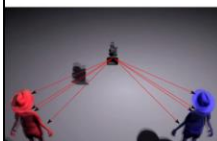
In this course, we will consider both:

- the problem of visibility determination (ie estimating how much a target is occluded)
- the problem of occlusion-free viewpoints determination (ie computing viewpoints from which target objects are visible)

For both problems, local and global techniques can be employed in similar ways:

- local techniques rely on a restricted knowledge of the environment (but can be easily updated)
- global techniques rely on a full knowledge of the visibility in the environment (that is expensive to update)

Ray-casting



- a technique of choice in most interactive camera control applications
 - low-cost: logarithmic in the number of objects
 - simple to implement
 - provided as a primitive in most interactive environments
 - targets/occluders can be easily abstracted by bounding volumes for performance
 - easy integration in the camera control pipeline
- but only provides a **rough and partial** estimation of visibility
 - most approaches evaluate a fixed number of rays (e.g. bounding box and center of the target)
 - requires to define a ratio between quality and cost



15

In ray casting approaches the candidate position for the camera is evaluated by casting a ray in the direction of the target object. An incremental improvement on simple ray casting approaches can be achieved by casting from an array of candidate camera locations (at a linear increase in cost), and, where the visibility of multiple target objects is required, by repeating the process for each target object. Deciding how to move the camera based on such collections of single point estimates of visibility has a number of limitations, for example, it is not possible to maintain partial visibility of a target object as it moves behind a sparse occluder (such as a set of railings). Furthermore, using a single point to approximate the geometrical complexity of a target object fails to sufficiently characterize its visibility.

Bounding volumes

- Principle:
 - both the targets and the camera are encompassed in a primitive shape (sphere, AA-box, OBB)
 - intersections are performed between the shape and the occluders (collision detection)
 - easy to encompass the temporal evolution of the camera/target in the primitive shape
- Sphere-based occlusion detection [CN05]

a

b

16

Using bounding volumes for visibility detection is a rough and conservative - yet rapid - way of estimating occlusion (i.e. can be used before more expensive techniques such as hardware rendering). Many libraries provide efficient means of detecting collision with primitives, and in most cases, the process only requires a boolean result from the test (ie. not the volume, depth or point of intersection). Courty & Marchand [CM01, MC02] avoid occlusion in a target tracking problem by computing an approximate bounding volume that encompasses both the camera and the target. Occluders (i.e. not the camera or the target objects) are prevented from entering the volume corresponding to target motion or camera motion. However, the approximate nature of the bounding volumes restricts both expressiveness (e.g. quantify partial occlusion) and practical application (e.g. over-estimation for complex shapes).

A study the evolution of the depth/volume of intersection is possible to get an idea of how occlusion is evolving. This can be used in a preventive way by proposing large volumes around the camera. However, these approximations are rough and the cost of computing the intersected volume may overshadow the lightweight advantages of the technique.

Estimating visibility (1)

Use hardware rendering techniques for estimating the visibility of a target object

- Step 1: perform a rendering of the target object
- Step 2: perform a rendering of the occluders
- Step 3: use a stencil buffer, occlusion queries, or pixel shaders to determine overlap between pixels
- even **low resolution** buffers provide a good estimation of the visibility
- visibility can be **weighted** by the informative value of parts of the targets
 - by rendering with an appropriate **importance map** (texture)
 - and computing the product of visibility and importance



17

Degree of visibility of the target is determined by the ratio between the number of visible pixels of the target and the total number of pixels of the target.

Increasing the resolution of the rendered buffers obviously improves the precision in the visibility estimation (and rapidly converges to a good estimation)

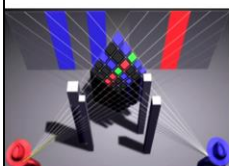
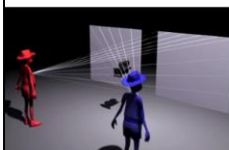
Occluders and target objects can have specific geometries adapted to the rendering

- low resolution geometries, partial models (eg remove arms and legs, keep hands and feet)
- removal of sparse occluders, or alpha blended textures (e.g. fine fences, leaves etc...)

Important regions on the surface of the targets can be either manually or automatically (silhouette, saliency) computed and rendered on the surface of

the target. Visibility can then be weighted by this importance map.

Occlusion-free views



Hardware rendering techniques for computing occlusion-free views:

- for a single target:
 - a rendering is performed from the target object to the area where visibility should be checked
 - only the depth information is stored and used (principle of shadow volumes)
- **locally**: projections can be performed in a small region of interest (around the camera configuration) [CON08]
- **globally**: projections can be performed all around the target
 - on the six faces of a bounding box [MMGK09]
 - projected onto the surface of a bounding sphere [Bares99]



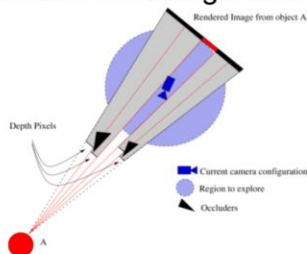
18

Only a small number of real-time approaches for occlusion-aware camera control have been proposed. Crucially, existing techniques (e.g [HO00]) cannot be easily extended to capture the full spatial extent of target objects (i.e. they model target objects as points). The computation of occlusion-free viewpoints is closely related to the well known problem of visibility determination [COCSD00, Dur00] which has a bearing on a range of sub-fields in computer graphics, from hidden surface removal and occlusion culling, to global illumination and image-based modeling and rendering.

Here we move from visibility estimation to the computation of occlusion-free viewpoints with hardware rendering techniques. The principle is close to the one of ray-casting: renderings are performed from the target object to the area where visibility should be checked, and most similar to the principles in shadow volume computation (studying the depth buffer to estimate whether the geometry is shadowed or not).

Occlusion-free views

- By rendering from the target surface towards a region of potential camera configurations:



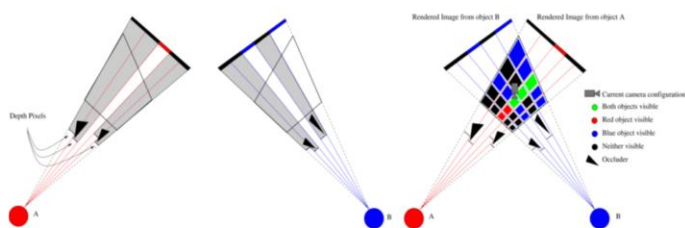
1. select a region to explore around the current camera
2. perform the rendering
3. study the depth-map for visibility



A clear parallel can be drawn between the problem of real-time soft shadow computation and real-time visibility computation of target objects. Target objects can be treated as light sources for which we need to compute the volumes outside of the shadow and penumbra (this is an inverse volume carving problem) in which to place a camera. One technique for real-time shadow computation relies on silhouette detection (e.g. penumbra wedges [AAM03]), that use the exact silhouette of objects to compute shadow volumes. However, the complexity of silhouette detection increases with the complexity the objects casting shadows and such approaches are also not readily applicable to rasterizable entities that use alpha-textures (which are increasingly used real-time 3D graphics). Another class of techniques that is used in camera control [CON08] relies on frame-buffer approaches that construct a depth map rendered from the location of light sources using graphics hardware. This shadow map is then sampled in relation to the world geometry and a simple depth comparison can be used the determine the status of a point in space (whether it is hidden by an occluder or not).

Visibility of multiple targets

- For multiple targets
 - composing visibility information: a task similar to composing multiple shadow maps (to form a penumbra map)
 - yet requires either a sampling process or the computation of the union of shadow volumes



In a given region, visibility for multiple targets (or multiple points on the target surface) is computed by performing one rendering per target. Depth information is composed in a way similar to penumbra maps (see next slide): the area is sampled and each sample is expressed in the local basis of each rendering in order to access the appropriate depth value in the shadow map. A specific way of composing depth maps is proposed in [CON08], where asymmetric frustums are computed for rendering. This technique avoids the sampling of the area by using a trilinear basis to access visibility information.

Discussion over local visibility techniques

- **simple** to implement and **efficient**
- CPU/GPU-adaptive (ray-casting or frame rendering)
- adapted to **dynamic** environments

But: lacks global visibility

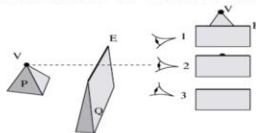
- leads to issues in local minima areas
- inappropriate for performing cuts between shots



The methods we reviewed provide efficient and CPU-adaptive approaches to locally establish visibility or compute occlusion-free views. However their intrinsic local nature prevent them from performing transition planning (moving from one viewpoint to another while maximizing visibility), and may fail in some situations (no local visibility). Furthermore, when cuts between viewpoints must be computed (eg. reverse shots), many local regions need to be sampled (with no guarantee of finding an appropriate view).

Global visibility techniques

- provides a collection of techniques and structures to represent the visibility in an environment:
 - grounded on the notion of *visual events*



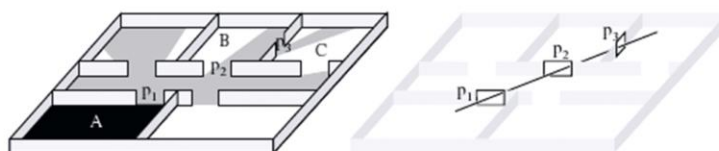
- a *visual event* separates the space into visible and non-visible areas
- two classes of problems are considered in the literature
 - *from-point* visibility computation
 - *from-region* visibility computation



Visibility methods aim to calculate either the regions of a space which can be seen from a point (from-point visibility computation), or those that can be seen from a region (from-region visibility computation). In simple terms, visibility determination uses visual events - the boundary configurations for which the visibility changes - to partition space. Such methods can be broadly categorized according to the space in which the partitioning is performed, that is, object space, image space, viewpoint space or line-space (for a detailed presentation see [Dur99]). Visibility methods in dynamic environments have mostly addressed the problem of updating these visibility representations for moving objects [SG99] and modeling moving occludees (e.g. motion volumes [DDTP00]).

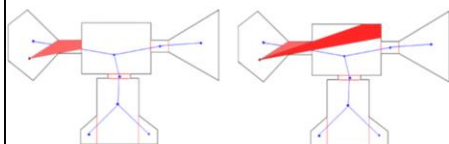
Cell-and-portal Visibility

- based on architectural environments:
 - scenes are subdivided into rooms (the **cells**)
 - visibility occurs through openings (the **portals**)
 - the cells are connected in an **adjacency** graph
 - each cell visibility is then established by propagating visibility in the adjacency graph and checking the portal-to-portal visibility

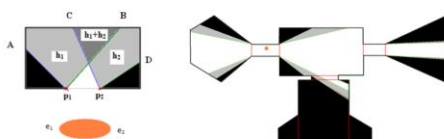


C&P visibility is restricted to architectural environments, though abstract 2D $\frac{1}{2}$ representations can be used to handle more complex scenes [Lam09]. C&P techniques have been initially proposed to improve occlusion culling in complex urban scenes (ie removing parts of the geometry that are hidden). The scene is decomposed into cells (or convex cells to ensure full visibility inside them – a constrained Delaunay triangulation helps to compute such a decomposition), and cells are connected by portals (which edges are the support for visibility). Inter-cell visibility propagation is then performed by constructing stabbing lines (lines that separate the visibility in space). Visible cells are connected together in an visible adjacency graph.

C&P for Target Visibility



From-point visibility



From-region visibility

- given the location of a dynamic target:
 - the adjacency graph is used to identify possible visibility areas
 - wrt. from-point visibility, the visibility of the target is established by using the **stabbing lines** defined by the portals
 - wrt. from-region visibility, the visibility can be established by bounding the region and using multiple **stabbing lines**
 - propagation is performed in the adjacency graph



Inter-cell visibility propagation is then performed by constructing stabbing lines (lines that separate the visibility in space). Visible cells are connected together in an visible adjacency graph.

Discussion

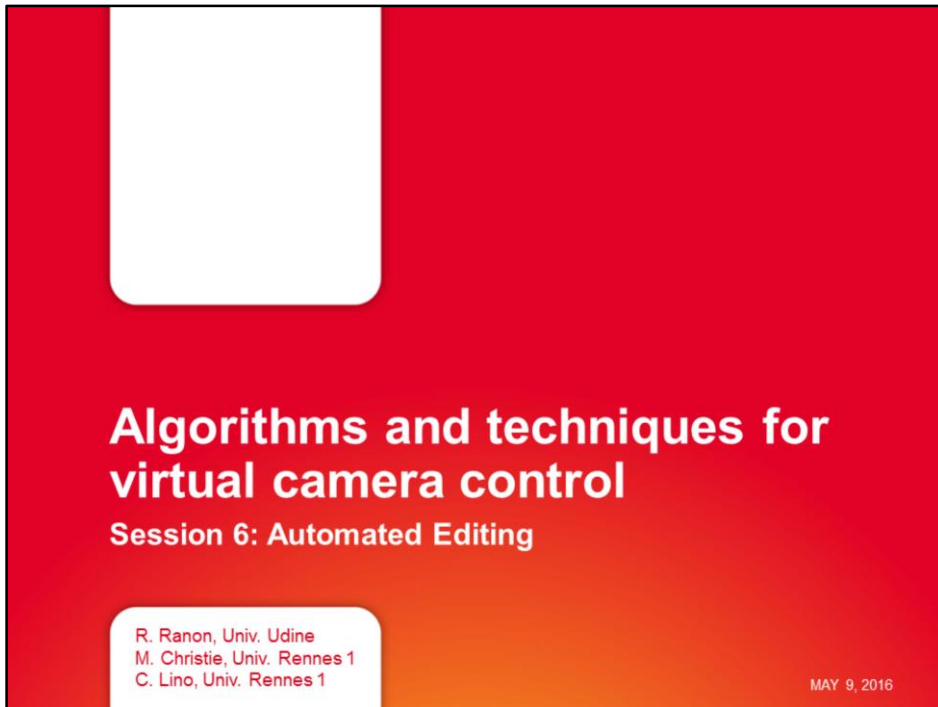
Handling visibility remains a complex topic:

- cost for precise/complete evaluation of visibility of complex/multiple targets
- strong link with planning techniques
- necessity of coupling of local and global visibility techniques
- importance of anticipating actions/motions
- importance of studying the nature of the targets and occluders



Bibliography

- [AVF04] Aandujar C. G., Vazquez P. P. A., Fairen M. G.: *Way-finder: Guided tours through complex walkthrough models*. Proceedings of the Eurographics Conference, 2004.
- [Bec02] Beckhaus S.: *Dynamic Potential Fields for Guided Exploration in Virtual Environments*. PhD thesis, University of Magdeburg, 2002.
- [HMK+ 97] Hong L., Muraki S., Kaufman A., Bartz D., He T.: *Virtual voyage: interactive navigation in the human colon*. In Proceedings of the Siggraph Conference, 1997.
- [NO03] Nieuwenhuisen D., Overmars M. H.: *Motion Planning for Camera Movements in Virtual Environments*. Tech. Rep, Utrecht University, 2003.
- [HHS01] Halper N., Helbing R., Strothotte T.: *A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence*. In Proc. of the Eurographics Conference, 2001.
- [LCF10] Lino, Christie, Lamarche, Schofield, Olivier: *A real-time cinematography system for interactive 3d environments*; Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer (SCA'2010)
- [OSTG09] Oskam, Sumner, Thuerey, Gross. 2009. "Visibility transition planning for dynamic camera control". In Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '09)
- [CDM14] Sanokho CB, Desoche C, Merabti B, Li TY, Christie M, "Camera Motion Graphs", Proceedings of the 2014 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '14)
- [LC15] Lino C. and Christie, M. "Intuitive and Efficient Camera Control with the Toric Space" Proceedings of SIGGRAPH 2015.
- [CNO12] Christie M, Normand JM, Olivier P, "Occlusion-free camera control for multiple targets", Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '12)



In this session we will adopt a more cinematographic view by studying how editing techniques from the cinema can be formalized in computational models to interactively or automatically generate well edited sequences of shots.

Let first have a look at how real cinematographers deal with the editing of a movie.

Introduction

- Film editing process
 - Shoot a set of rushes
 - Cut / paste pieces of rushes
 - Create a whole storyline
- Tedious and skillfull process
- Visual « grammar »: **Continuity-editing**
 - *Grammar of the Film Language* [Arijon 76]
 - *Grammar of the Shots* [Thomson 98]
 - *Grammar of the Edit* [Thomson 93]
 - *The five C's of Cinematography* [Mascelli 98]
 - ...



The work of the editor of real movie is a to take as input the rushes that have been shot, then to diligently cut and paste pieces of those rushes to create a whole storyline as output.

This is a tedious and technical task, and the cinema industry has thus been building a “visual grammar” (aka continuity-editing) of how to properly shoot and edit movies for more than a century.

One can find a number of well-known “cook-books” each providing a set of practical or theoretical rules that allow selecting well-composed shots that can properly convey movie actions and cuts that can enforce some continuity in actions through the sequence of shots.

Shots grammar: « action »

- Controls how viewers are focused on the relevant actions of the storyline
 - Should be **informative enough**
 - Should **highlight** important information (provide guidance to viewers) and **avoid distracting viewers**



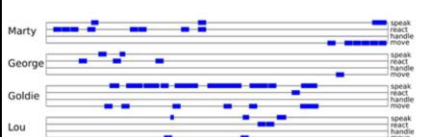
Walk action



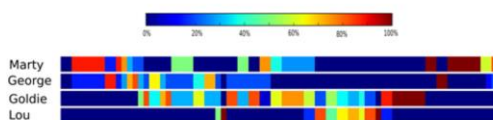
Speak / React action



Manipulate action



Actions unfolding over time



Importance of characters over time



First, they provide a grammar of the shot. Selected shots where enough space is left to perform the action. For instance for a shot show a character walking, enough screen space should be left in his motion direction, if showing a character speaking or reacting after another character has been speaking, the shot should provide some look-room or head-room to the character (i.e. leave enough space in his gaze direction).

Then protagonists should be highlighted so as to best highlight the main actions that are unfolding at that moment in the story.

Cuts grammar: « continuity » of actions

- Controls how storyline actions are perceived all together
 - **Make link** between pieces of information
 - **Guide** viewers' attention (visual cues)
- Controls how a given action is perceived as continuous in time
 - **Do no break continuity** (coherency)

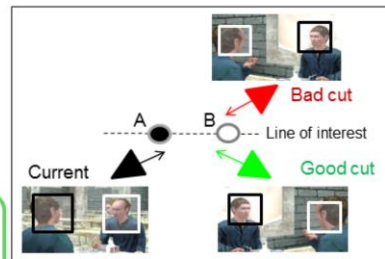
Jump Cuts



Continuity errors



Keeps continuity



Then, they provide a grammar of the edit. Selected cuts between two shots should be invisible to viewers. And to do so, they need not to break the visual continuity of the actions.

Most important continuity rules are that a cut should not provide too much change on the on-screen position of a character that the viewer is looking at, as it will force the viewer's eye to move to the new position after the cut (after a few cuts it can thus lead to some visual discomfort). The cut should also maintain relative positions of objects on the screen. It should enforce the continuity in the motions of characters (a character moving from left to right in one shot, should keep moving in that direction in the following shot), and it is the same for their gaze directions (a character looking to the right should keep looking to the right). Finally, a cut should be perceived as a cut, i.e. it should provide sufficient change on the characters on-screen size and/or on their view angle (in other cases, it is known as a "jump-cut", which is perceived as fast camera motion instead of a cut).

Overall pace of the story

- Controls how the story actions are perceived / understood
 - Pace is too fast
 - **Not easy to understand** what is occurring
 - May introduce visual discomfort after a while !
 - Pace is too slow
 - **Information redundancy**
 - May become boring to watch after a while !
- Requires a « **natural** » distribution of shots durations



Fast pace (durations around 2 seconds)



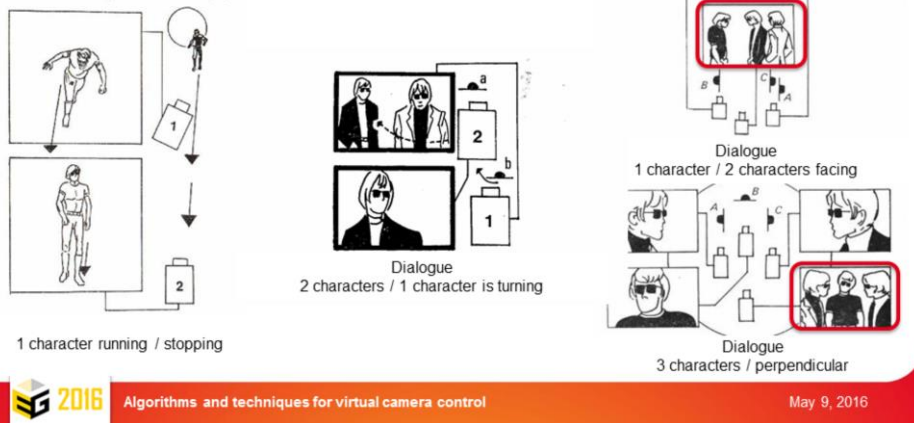
Slow pace (durations around 10 seconds)



The durations of shots is an also important criteria for editing movies, as it will control the pace of the story. If the pace is too fast (i.e. shots are too short) it does not lets enough time to viewers to “read” the content of the shot. If the pace is too slow (i.e. shots are too long), it lets too much time to the viewer to “read” the whole content of the shots (comprising background actions or landscape) so it can become boring for viewers to watch a shot after a while. Obviously the editor instead has to make a compromise, depending on actions complexity (how much information the director would like to provide to viewers) and the rhythm of actions. This should therefore lead to “natural” distribution of shots durations (i.e. if cutting at regular intervals of time, the viewer would be able to perceive cuts).

Shooting and editing using « Cook-books »

- Example of Arijon's film grammar
 - On a case-by-case basis
 - Depends on the number of characters
 - And the type of action (static / moving characters)



To account for such theoretical rules, some « cook-books » such as the one of Daniel Arijon (which is surely the more cited in the literature), are providing more practical rules on how to place camera to shoot the actions and how to cut between them along time.

In his book, Arijon is providing rules on a case-by-case basis.

For instance, to shoot a single character moving, you can place two cameras in front of the character, at the start and end of his motion, ensuring that cameras are located on the same side of a line defining his motion.

To shoot two or three characters talking to each other one can position cameras on one side of the line of interest drawn through characters, then depending on the configuration and motions or characters in the story, it will require more or less cameras to be placed around characters and their placements will be slightly different.

More generally, one can find a configuration of camera for each kind of action, and each number and configuration of characters.

Naive approach to automated editing [He et al. 96]

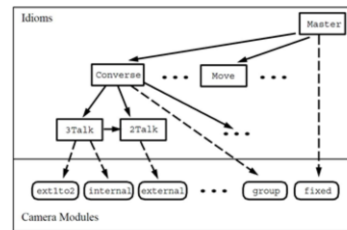
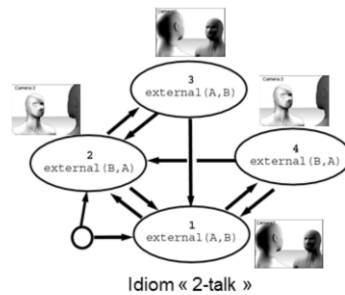
- Encode idioms
 - Stereotypical way of shooting an action
- Idioms = finite state machines (FSM)
 - New type of action \Rightarrow new FSM
 - New number of targets \Rightarrow new FSM

😊 Can run in real-time

⊗ Requires as many FSM as cases

⊗ Not flexible (cannot provide variations)

⊗ Hard to generalize !



In handling the automated editing in 3D environments, one can easily encode continuity rules by following cookbook.

A “naive” approach proposed by He in his 1996 SIGGRAPH paper is to encode idioms (stereotypical ways of filming a given action, as provided by Arijon) as a finite state machine. In this FSM, a node will represent a single viewpoint, and an arc will represent a possible transition between two given viewpoints (i.e. a cut or camera motion). Transitions can then be parametrized (i.e. cut when first character start speaking, or after 10 seconds spent in the shot).

Still following this way of implementing cook-books, one has to encode as many FSM as the number of different types of actions and number of characters. Then, one can also define a tree of those idioms which will handle the transitions between the multiple actions performed in the story.

The great advantage of this model is that it can be easily implemented and that it can run in real-time. Therefore it remains one of the preferred models in computer games today.


There have also been a large number of scientific papers build upon this idiom-based idea to create more evolved editing systems.

But, the big problem with this model it is hard to generalize it since it is a bit too rigid.

Today we will introduce a more general way of formulating and implmenting the editing process.


Better formulation for continuity-editing

- Reproduce the editing process
- Start from a set of rushes
- **Rank possible edits**
 - **Quality of Shots**
⇒ Action cost
 - **Quality of Cuts**
⇒ Transition cost
 - **Quality of Pace**
⇒ Rhythm cost
- **Choose the best edit**
 - Combine costs into an objective function to minimize




Poor shot **Poor shot**

Bad quality of shots
+ **Good** quality of cuts (no discontinuity)
⇒ **Very low quality edit**




Left-to-right discontinuity **Gaze** discontinuity

Good quality of shots
+ **Bad** quality of cuts (some discontinuities)
⇒ **Medium quality edit**



Good quality of shots
+ **Good** quality of cuts (continuity)
⇒ **High quality edit**



Algorithms and techniques for virtual camera control

May 9, 2016

Here we are targeting to better reproduce the editing process as it is done by a real editor.

We start from the set of input rushes and we want to provide the best edit possible as output.

Our automated process will then be divided into two steps: (i) provide a way to evaluate the quality of a given edit using these rushes, then (ii) explore the range of possible edits and choose the one which obtains the best evaluation.

In the first step (evaluating an edit) we split the overall quality of the edit into three components: the quality of shots (which is really important as poor shots will lead to a very bad edit), the quality of cuts (which should enforce continuity), and the quality of pace. And we will consider that a good edit is obtained when all three components are evaluated as good.

To do so, as previously, we can derive a cost function for each component, build an objective function aggregating these costs and try to minimize this objective function. We will see how to search for the best edit a bit later,, for now let's focus on each component separately.

Action cost (Shot quality)

- Penalize shots that do not convey enough of the important actions or distract viewers

- Visibility of actions
 - Important characters
 - « meaningful » body parts w.r.t. actions

- Cost function:

$$P_{Vis}^A(c_j^i) = \sum_i \sum_{part} \left[Imp(T_{part}^i, t) \cdot \frac{Occl(T_{part}^i, c_j^i)}{Size(T_{part}^i, c_j^i)} \right]$$

Convey
the actions

Distract
from
the actions



A first element in making a good shot is that it should convey enough of the relevant actions unfolding at that time in the story, and avoid distracting the viewer from the main story elements.

For example, to shoot a given action, this action should be fully visible in the frame. Which means that the protagonists of this actions should be visible on the screen, and more particularly their relevant body parts (those participating in the action). For instance here we have a character speaking so we would like his head to be visible on the screen.

To evaluate how much this rule is enforced, we can compute the area covered by the face of the character and compare it to the area it would cover if it were not occluded at all.

Then, following that principle we can build a cost function that will sum up over all body parts of all targets that appear on the screen. And we can also weight each character with regards to its importance in the story (i.e. how much it is participating to the unfolding actions) to penalize occlusions of protagonists more than secondary or background characters.

Action cost (Shot quality)

- Penalize shots that do not convey enough of the important actions or distract viewers

- Hitchcock rule:

- the more important the character
- the more it should fill the frame

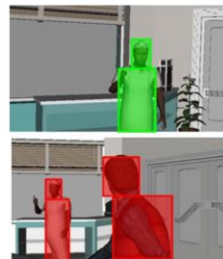
- Cost function:

$$P_{Hitch}^A(c_j^t) = \sum_t \left[\frac{Imp(T^i, t)}{\sum_k Imp(T^k, t)} - \frac{Size(T^i, t)}{\sum_k Size(T^k, t)} \right]$$

Convey
the actions

Distract
from
the actions

Actions fill the frame

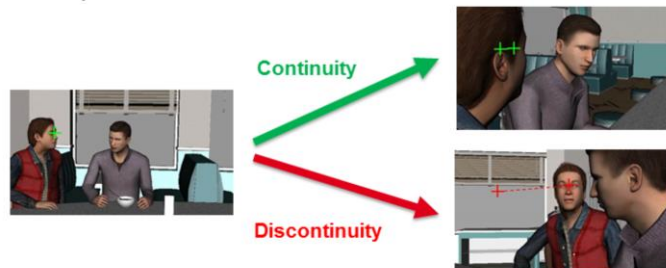


In the same way, a rule which was followed by Hitchcock and that we find really practical is that the more an action or a character is important in the story, the more it should fill the screen space (e.g. if only one action is unfolding, then only this action should be frame). Here we have a character talking to himself at one side of the scene, so the character should ideally fill the frame, i.e. he should be alone on the screen.

To evaluate how much this rule is enforced, we can compute the area covered by a character on the screen and compare it to the total area covered by all characters. Then, we can build a cost function that will compare the relative importance of a each character at that moment to the relative amount of the screen it fills (compared to other all characters).

Transition cost (Cut quality)

- Penalize cuts breaking continuity
 - On absolute screen positions



- Cost function:

$$P_{Screen}^T(c_{j-1}^t, c_j^t) = \sum_i \phi_S[Pos(T^t, c_{j-1}^t) - Pos(T^t, c_j^t)]$$

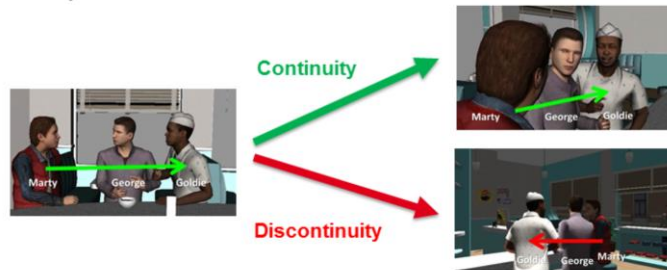


Now, if we look at the quality of a cut, we can consider a range of continuity-editing rules and derive a cost function to evaluate how much each is respected.

For example, we said that one should enforce (as much as possible) the absolute on-screen positions of characters. So, we can first compute the 2D screen position of a character before and after the cut, and compare both positions (the greater the on-screen distance, the greater the cost). We can then derive a cost function that will sum up the change of on-screen position for all characters appearing both before and after the cut.

Transition cost (Cut quality)

- Penalize cuts breaking continuity
 - On relative screen positions



- Cost function:

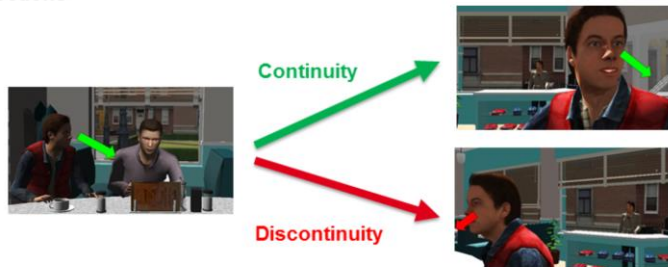
$$P_{Order}^T(c_{j-1}^t, c_j^t) = \sum_{i,j} \phi_o[Order(T^t, T^j, c_{j-1}^t), Order(T^t, T^j, c_j^t)]$$



In the same way, one should also maintain relative position of characters on the screen. To encode this rule, we can use the computed 2D screen positions of each character (before and after the cut), and compare the relative positions for each pair of character before and after the cut (we penalize when positions are reversed). We can finally derive a cost function that will sum up these penalties over all pairs of characters on the screen.

Transition cost (Cut quality)

- Penalize cuts breaking continuity
 - On gaze directions



- Cost function:

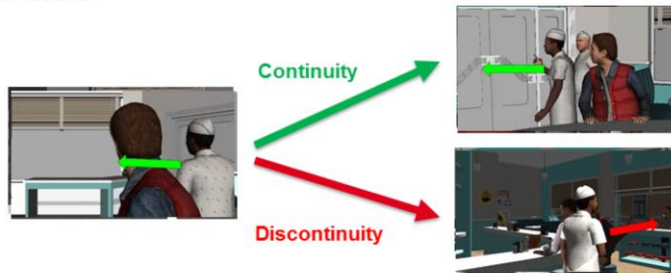
$$P_{Gaze}^T(c_{j-1}^t, c_j^t) = \sum_i \phi_G[Gaze(T^i, c_{j-1}^t), Gaze(T^i, c_j^t)]$$



Another rule is to enforce continuity on characters' gaze. To encode this rule, we can compute the projection of the gaze of a character (before and after the cut), and compare both directions (we penalize when gaze directing is changing, for instance if the character was looking left before the cut and is looking right after the cut). We then derive a cost function that will sum up these penalties over all characters appearing on the screen.

Transition cost (Cut quality)

- Penalize cuts breaking continuity
 - On apparent motions



- Cost function:

$$P_{Motion}^T(c_{j-1}^t, c_j^t) = \sum_i \phi_M[Motion(T^i, c_{j-1}^t), Motion(T^i, c_j^t)]$$



A similar rule is to enforce continuity on the apparent motion of characters (when they are in motion). To encode this rule, we can compute the projection of the character's velocity vector (before and after the cut), and compare both vectors in the screen space (we penalize when motion directing is changing, for instance if the character was moving to the left before the cut and is moving to the right after the cut). We then derive a cost function that will sum up these penalties over all characters appearing on the screen.

Transition cost (Cut quality)

- Penalize cuts that do not look like cuts (visually, not enough change in size or view angle)
- Jump cuts



Sufficient change in size



Sufficient change in view angle



« Jump cut »

- Cost function :

$$P_{Jump}^T(c_{j-1}^t, c_j^t) = \sum_t \phi_j[\Delta Size(T^t, c_{j-1}^t, c_j^t), \Delta Angle(T^t, c_{j-1}^t, c_j^t)]$$



Finally, we want to avoid jump-cuts, i.e. provide either a sufficient change in the screen size of characters or a sufficient change in the characters' view angle (or both). We could split this problem into two separate constraint that we could enforce separately but this would lead to force the editing system to enforce both at the same time, which would prevent some « grammatically correct » edits (such as those on the left figure) to be considered as good.

To improve, we can instead compute the change on each feature separately then combine them into a single cost function.

We first compute the on-screen size before and after the cut, and build a delta-size function returning a satisfaction value corresponding to how much the size change is sufficient.

In the way, we can build a delta-angle function returning a satisfaction value corresponding to how much the view angle change is sufficient.

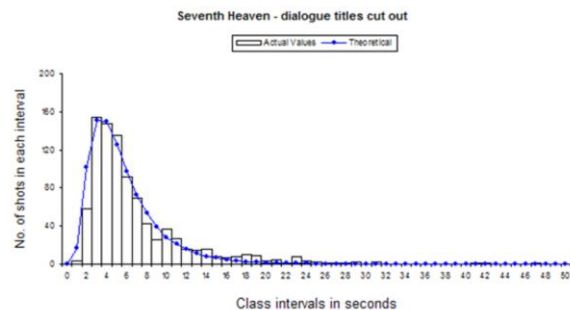
Finally we can build a cost function taking these two delta functions and penalizing cases where neither is sufficiently satisfied. We can finally sum up these penalties over all characters appearing on the screen both before and after the cut.

Rhythm cost (Pace quality)

- Film edits commonly follow a log-normal distribution of shots durations [Salt 03]
 - Provide the parameters of the log-normal distribution: mean (μ), stddev (σ)
 - Try to fit the durations d_j of all shots j with this distribution

- Cost function:

$$p^R(d_j) = \frac{(\log d_j - \log \mu)^2}{2\sigma^2}$$



In term of evaluation of the overall cutting rhythm, one can note that studies of the pace in real movies have shown that shots lengths tend to follow a log-normal distribution. Which means that most of the shots will have a duration close to a mean duration, some of them will be a bit shorter or longer, and very few of them will be much longer.

Following that idea, what we have proposed is to rely on the provision of a given pace by the user, through both parameters of a log-normal law (the mean shot duration and the standard deviation of duration from the mean value). The editing system should then build an edit that fits such a distribution as much as possible.

Here we evaluate how much a shot follows the log-normal law by computing the deviation of its duration to the expected mean duration and comparing the result with the expected standard deviation.

Searching for a good edit

- Should minimize the amount of editing errors
 - Find a « grammatically-correct » edit
- Means:
 - Convey relevant actions
 - Avoid discontinuities when cutting
 - Apply an appropriate cutting rhythm
- Build an objective function to minimize:

$$P(s) = W^A \cdot \sum_j \sum_{t_j \leq t \leq t_j + d_j} P^A(c_j^t) + W^T \cdot \sum_{1 \leq j} P^T(c_{j-1}^t, c_j^t) + W^R \cdot \sum_j P^R(d_j)$$

Action cost
(Shot quality)
Transition cost
(Cut quality)
Rhythm cost
(Pace quality)

Where $s = \{s_0, s_1, \dots, s_{j-1}, s_j, \dots, s_n\}$ is a sequence of shots (edit)



Well, we have shown how we can evaluate all three components, and rank the satisfaction of the related rules, separately.

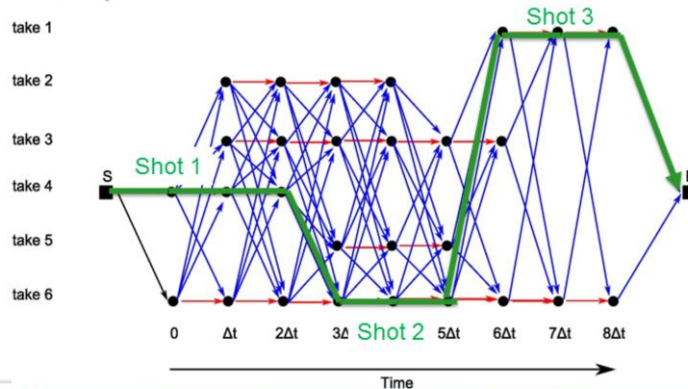
Now, we consider evaluating a full edit, and searching for a good one. What is important here is that a good edit should make as few errors as possible (we call it a « grammatically correct » movie).

This means that the edit should convey as much as possible of the actions, avoid discontinuities when cutting, while following an appropriate cutting rhythm.

We formulate that problem through an objective function to minimize. This objective function is built as a weighted sum of all three components costs, which in turn are made of all costs related to the rules we have presented.

Editing graph

- Automated editing can be viewed as planning a path through an oriented graph
 - Node c_i^t : use camera (take) i at time t
 - Arc $c_i^t \rightarrow c_j^{t+1}$: do not cut ($j = i$) / cut to another camera ($j \neq i$)



Now we can go into the process of searching the best edit.

If we consider we won't make time ellipses, one can view this search as choosing which camera to use at each time frame of the storyline.

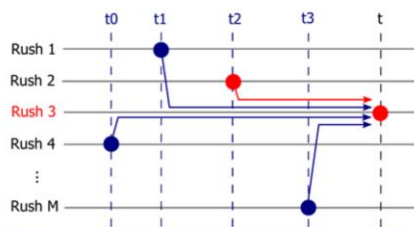
We can thus represent the space of all possible edits as a directed graph (with arcs going from left to right) where a node represent the use of a given camera at a given time frame, and an arc represent the transition from a camera at time t to a camera at time $t+1$. One can also consider that the graph is complete (i.e. one can continue using the current camera or cut to any other camera between each time frame).

This means that, taking one possible edit (i.e. a given path through this graph) from the beginning to the end of the story, we can now evaluate the edit using the previous objective function.

Finding the best edit thus requires evaluating all the possible paths, but this is however computationally too expensive.

Dealing with computational complexity

- A huge amount of possible paths through the editing graph
 - With M rushes and N frames \Rightarrow Requires to evaluate M^N edits !
 - Space complexity is also M^N
- Semi-Markov assumption on the editing process
 - Cut after c_i^t : decision depends only on the current shot (from last cut)
- Rely on a dynamic programming algorithm
 - With M rushes and N frames \Rightarrow complexity becomes M^2N^2 in time (and MN in space)



Indeed, if we consider that we have M rushes (cameras) over N time frames, this leads to a huge complexity both in computation time and in required memory space; which for now makes solving our problem impracticable.

To reduce this complexity, what is important here is that we can make a strong assumption on the editing process.

Actually, deciding whether or not to cut to another shot is only dependent on the amount of time we have already spent in the current shot, which allows some reduction on the search process.

Finally, building on this semi-Markov assumption, we have used a dynamic programming algorithm. Dynamic programming is really well-suited to this type of cases, where one can decompose a problem into a set of simpler sub-problems, that can in turn be solved separately from each other.

In our case, this allowed drastically reducing the complexity both in computation time and in memory space required (to encode the editing graph), making the search process more practicable.

Bibliography

- [HCS 96] He L., Cohen M. Salesin D. *The virtual cinematographer: a paradigm for automatic real-time camera control and directing*. In Proc. of the Siggraph conference, 1996.
- [CAH+96] Christianson D., Anderson S., He L., Salesin D., Weld D., Cohen M. *Declarative camera Control for Automatic Cinematography*. In Proc. of the AAAI conference, 1996.
- [ER07] Elson D., Riedl M. *A Lightweight Intelligent Virtual Cinematography System for Machinima Production*. In Proc. of Conference on Artificial Intelligence and Interactive Digital Entertainment, 2007.
- [KC08] Kardan K., Casanova H. *Virtual Cinematography of Group Scenes using Hierarchical Lines of Actions*. In Proc. of ACM SIGGRAPH Symposium on Video Games, 2008.
- [KC09] Kardan K., Casanova H. *Heuristics for Continuity Editing of Cinematic Computer Graphics Scenes*. In Proc. of ACM SIGGRAPH Symposium on Video Games, 2009.
- [LCL+10] Lino C., Christie M. Lamarche F., Schofield G., Ollivier P. *A Real-time Cinematography System for Interactive 3D Environments*. In Proc. of ACM SIGGRAPH / Eurographics Symposium on Computer Animation, 2010.
- [GRL+15] Galvane Q., Ronfard R., Lino C., Christie M. *Continuity-editing for 3D Animation*. In Proc. of AAAI Conference on Artificial Intelligence, 2015.





inria
Informatiques mathématiques

Algorithms and techniques for virtual camera control

Session 7: Applications, Trends, Issues

M. Christie, Univ. Rennes 1
C. Lino, Univ. Rennes 1
R. Ranon, Univ. Udine

Trends

- more practical impact of the research field
- data-driven camera control (taking inspiration from real movies)
 - camera trajectories
 - movie style
- application to more complex problems



In the last years, we are witnessing the first applications of sophisticated camera control algorithms to real-world applications. For example, SolidFrame (<http://www.solid-frame.com>) is a new tool designed for animators and cinematographer to interactively explore a large collection of shots over their 3D animation, and rapidly create and compare multiple edits of the

same animation; VEX-CMS

(<http://hcilab.uniud.it/vex/>) is an application that allows people with no expertise in 3D modeling to create 3D virtual exhibitions and virtual visits integrating objects and information into virtual environments.

Another interesting trend is the exploitation of data from real-world movies, e.g. to generate better camera compositions and trajectories. Recent work done at INRIA France (only partly published) proposes methods to reproduce camera compositions taken from real movie shots, and to reuse camera trajectories taken from real movies (also explored in [Kurz et al 2010]), in an effort to provide more believable and natural camera movements.

We are also seeing camera control research papers that challenge problems that are significantly more complex than the "toy" problems that were addressed by earlier papers in the field. For example, [Ranon et al 2015] applies some of the camera control algorithms we have seen in this course to the problem of visualizing results from aviation safety simulation scenarios that involve hundreds of characters and highly frequent events; [Galvane et al 2013] propose camera control algorithms to film scenes involving a crowd of characters.

Issues

- computing visibility
 - practical models to handle the full visual extend of targets
 - temporal vs spatial visibility
 - visibility is related to perception, and ultimately to recognition
 - integration into planning techniques
- take lighting into consideration
 - a fundamental information for proper composition in photography and cinematography



Visibility is actually one of the main issues in camera control, and has been quite neglected. With the advent of efficient dedicated graphical languages, such issues are currently re-explored.

Visibility is not only difficult from a technical point of view, it also is related to more fundamental aspects in perception that are critical to evaluate

(recognizability, task-dependent, duration and extent of the occlusion).

Directions for research

- more expressivity (requirements)
 - especially considering aesthetics, cognitive aspects, conveyance of narrative elements and emotions
- integration cognitive and perceptual models into automated editing
- coupling staging, lighting, and cameras
- adapt camera control techniques to control real cameras (drones, automated camera cranes, ...)

References

- [Kurz et al 2010] Christian Kurz, Tobias Ritschel, Elmar Eisemann, Thorsten Thormahlen, and H-P Seidel. Camera motion style transfer. In *Visual Media Production (CVMP)*, 201, pages 9–16. IEEE, 2010. 52, 90
- [Ranon et al 2015] Ranon R., Chittaro L., Buttussi F., Automatic Camera Control meets Emergency Simulations: an Application to Aviation Safety, *Computers and Graphics*, Vol. 48, May 2015, pp. 23–34
- Galvane, Q., Christie, M., Ronfard, R., Lim, C.-K., AND Cani, M.-P. 2013. Steering Behaviors for Autonomous Cameras. In *Motion in Games*, ACM, 93–102.