EG UK Theory and Practice of Computer Graphics (2011)
Hamish Carr, Ian Grimstead (Editors)

A Desktop Multi-Touch Interface for Posing Characters

Ian Stephenson[†]

National Centre for Computer Animation
Bournemouth University

Abstract

Multi-touch, direct manipulation interfaces have become common in mobile media applications, but their use on the desktop is limited. In this paper we propose that multi touch direct manipulation is well suited to the posing of animated characters using inverse kinematics, and demonstrate how it can be implemented in a desktop application, by the addition of commodity mobile touch devices. We also describe how the multi-touch display software developed for this system can be integrated into a number of other applications.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware Architecture—Input Devices. I.3.6 [Computer Graphics]: Methodologies and Techniques—Interaction Techniques. I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation.

1. Introduction

The posing of human and other characters is one of the most challenging tasks in computer animation, and is generally tackled using a combination of forward kinematics (FK: where joint rotations are set) and inverse kinematics (IK: where the position of key bones is specified, and the rotations of the joints is automatically calculated to realise this) [Wel89]. While IK is conceptually simpler for the end user, it can be difficult to control and the results can be unpredictable — when a hand is moved the elbow may move to an undesirable position. As a result users can find IK frustrating and it is usually necessary to use a combination of IK and FK to obtain the desired results [Kar02].

We planned to incorporate the posing of human characters into a much larger design application for users who were not trained as animators, and as such existing IK, FK and hybrid approaches were considered inappropriate. Posing characters is only a small part of the users' goals. and as such the interfaces of existing tools such as Poser which is dedicated to the posing of characters, presented a level of confusion and scope for error that was unsuitable for our target users. While the precision required by film animators was not a requirement of our system, it needed to be intuitive and robust.

Observing drama and dance teachers as they correct the pose of students we note that they are essentially forced to use an IK approach — they move a part of the student's body to the desired position. Despite this they are able to pose their "characters" quickly and reliably without FK. In part this is due to the students cooperation (the student effectively acting as an IK solver with embedded domain knowledge) and that they are working directly in the 3D environment rather than on screen. However an important advantage they have over computer animators is that they can manipulate two parts of the students body at once — for example consider a ballet teacher adjusting the position of a students foot but holding her knee to prevent it moving. This avoids exactly the kind of undesirable results which typically arise during IK manipulation.

Based on this observation we propose the use of a multi-touch direct manipulation interface for posing of characters using IK. However while suitable multi-touch hardware is common in mobile media applications it is rare to find such hardware on the desktop where it could be integrated into an animation tool or pipeline. We therefore also propose the use of a commodity media device as a peripheral to a standard computer system, acting as both an auxiliary display and touch input device. We will first describe the implementation of the multi-touch input device, and demonstrate its use on a number of projects, then consider its application to the problem of posing characters.

[†] e-mail:ian@dctsystems.co.uk

DOI: 10.2312/LocalChapterEvents/TPCG/TPCG11/089-095

2. Multi-Touch Desktop I/O

2.1. Existing Technology

Touch screen displays which resemble the standard display on desktop computer systems are common in point of sale applications, and other systems where an untrained user is interacting with a device to perform a specific task of short duration. While experiences with such devices make touch interaction appear an attractive addition to regular computer displays, experiments quickly show that such screens are impractical for precision work over extended periods of time, due to their physical dimensions and orientation [ALM92]. Users become fatigued by reaching out to the vertical screen, and the large movements required to interact with objects over the area of typical computer screens.

The most common form of touch input is found on laptop computers takes the form of a trackpad. While ergonomically viable, trackpads lack the visual feedback required for true direct manipulation. Some trackpads support multi-touch input, but without the ability to relate a specific touchpad location directly to a unique point on screen, they are only suitable for gesture input [HSH*05]. A number of graphics tablets are available which combine pen input with a display, but these are expensive, and do not support multi-touch.

In contrast handheld media devices such as mobile phones and mp3 players have successfully incorporated multi-touch screens due to their small size. This enables the user to position them comfortably, interact directly over the whole screen area with only small movements, and allows them to be produced at low cost. However the limited screen size, and processing power of such devices make them an unsuitable platform for 3D animation packages. Even if such software could be developed within the constraints of a mobile device, it would be difficult to integrate within a production pipeline. Developing and distributing software for mobile devices can be subject to licensing restrictions [App11], making it difficult to create and maintain software in the style necessary for animation production.

2.2. ScreenPad

By utilising a multi-touch media device (specifically an iPod Touch, iPad or iPhone) as a peripheral to a standard desktop computer it is possible to combine the strengths of each. The main application is hosted on a computer with a standard on-screen UI, but the iOS device provides a secondary screen, which presents a multi-touch interface focused on a specific part of the task. The host application can make use of the full resources of the desktop machine, including traditional I/O methods, but draw upon the mobile devices multi-touch screen as required.

Mobile applications currently exist which allow an iOS device to act as a remote control for specific pieces of software (most notably Apple has developed apps to control iTunes and Keynote), but these act in place of traditional hardware remote controls. They make limited use of the touch devices capabilities, and are tied to controlling only one application, such that even minor updates to the desktop software require the remote to updated. Such paired applications are difficult to develop and maintain, as experience of two development environments is required, and the two applications must be developed in parallel.

A second set of mobile applications attempt a more generic interface to the desktop system, providing a remote desktop facility (through VNC, or Windows Remote Desktop Services) which mirrors the desktop machine's screen on the mobile device, and uses touch input to control the mouse pointer. While potentially useful when access to the host machine is otherwise unavailable, the small screen size makes these of limited practical use. Additionally the desktop software has no knowledge of the touch devices existence so is unable to make use of its additional features over a regular pointing device. Applications such as "Mobile Mouse" use the mobile device for a remote for the OS itself, allowing the desktop to be controlled from a device in a convenient fashion, but do not essentially extend the desktop paradigm.

Between the extremes of a custom remote control, and a virtual desktop, we propose a generic solution akin to a thin client, which allows a desktop application to draw freely on the remote screen, and receive touches from it, essentially extending the desktop application onto the touch screen. As the desktop application is customised to support the remote input it can make full use of the multi-touch facilities, but without the need for the developer to write code to run on the mobile device.

As only a single generic application need be installed on the touch device for any number of desktop applications, the developer of the host application does not need to be familiar with iOS, or the associated issues of licensing and distribution. All the application specific code is contained on the host, so there is no need to upgrade the mobile devices software as the host application is developed.

The system provides an interface from the host to the resources of the mobile device, and as such imposes no UI constraints. However the interfaces so far developed have been of the form of a standard desktop application, with one window displayed on the touch device to receive touch input. This window may or may not be visible on the main desktop. In some respects this is similar to the mechanism on Sega Dreamcast games console [HO99] where each controller incorporated a small display, allowing a game to present unique private information to each player. While recognised as having great potential the system was discontinued before this could be realised. This concept has been revived in the recently announced Nintendo Wii U [Nin11], though this system is not yet commercially available.

```
@interface ScreenPadClient : NSObject {...}
-(ScreenPadClient *)initWithDelegate:(id<ScreenPadClientDelegate>)target;
-(void) sendImage:(NSImage *)image;
-(NSArray *)currentTouches;
-(NSSize)viewSize;
-(float)viewScale;
-(int)viewOrientation;
@end

@protocol ScreenPadClientDelegate
-(void)screenPad:(ScreenPadClient *)sp didConnectToService:(NSNetService *)service;
-(void)screenPadDidDisconnect:(ScreenPadClient *)sp;
-(void)screenPad:(ScreenPadClient *)sp didReceiveTouches:(NSArray *)touches;
-(void)screenPadDidReceiveInfo:(ScreenPadClient *)sp;
@end
```

Figure 1: *The ScreenPad API*

2.3. API

Support for the ScreenPad interface can be added to a desktop application using the API shown in figure 1. This is implemented using the MacOS Objective C development environment, but source code is provided and could be ported to other platforms.

The desktop application (which is technically a client of the ScreenPad service), creates an instance of the `ScreenPadClient` object, using the `initWithDelegate:` method. This uses mDNS zeroconf discovery [SC05] to establish a connection to a touch screen, and reports a successful connection via the `screenPad:didConnectToService:` delegate method. There is no authentication built into this level of connection, so it is essentially insecure, but additional security could be built into a higher layer, essentially requiring the touch device user to verify that they are also the desktop user. More easily, a private ad-hoc wifi network can be created by the desktop machine, to which only the trusted device has access. This has the additional benefit of ensuring good wifi performance between the two machines.

Once a connection has been established the desktop application can update the touchpad screen by simply sending images to it using the `sendImage:` method. Though potentially inefficient (the whole screen image must be sent for each update), this creates the most simple, but flexible drawing process as the image can be imported from a file, drawn using standard MacOS drawing functions, or retrieved from the OpenGL frame buffer. Additional methods are available which allow control of the format and compression used when an image is transmitted. Images are automatically buffered, and frames dropped to ensure that good performance is achieved on mobile devices with a range of network and cpu performance.

The host application can obtain information about the touch device using the `viewSize`, `viewOrientation` and `viewScale` methods, though these are not required as the image is automatically scaled to fit the display. Should the orientation of the device change the delegate's `screenPadDidReceiveInfo:` method is called allowing it to react to the change.

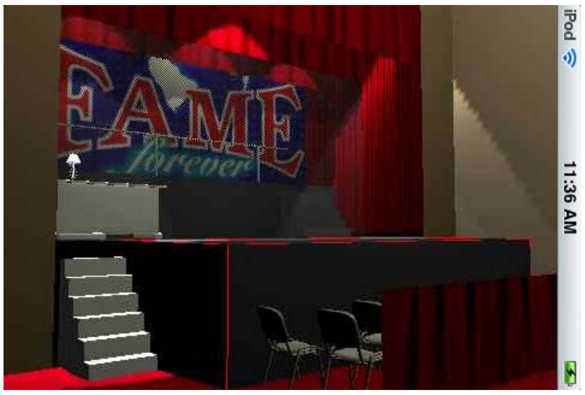

Figure 2: *StageBuilder displayed on an an iPod Touch.*

Information on user touches is provided through the `screenPad:didReceiveTouches:` method which provides an array of the currently active touches whenever the touch state changes.

2.4. Applications

The ScreenPad client API has been used in a number of applications, of varying complexity. As a result of this it has developed into a robust, and flexible tool, which is simple enough to be integrated into existing applications quickly, while powerful enough to allow developers to build the interface they require.

The simplest form of application using ScreenPad is one which simply pushes images to the remote display. An example of this would be an application allows an iOS device to display the view from a desktop machines webcam. The desktop machine creates a client object, and once connected it grabs an image from the webcam, rotates it to the correct orientation and pushes it to the touch device. When the image has been successfully received by the touchscreen, it sends an acknowledgement, which triggers the sending of the next image. Multiple clients can be connected simultaneously.

StageBuilder [SP10] is a large application previously developed for the modelling of theatrical stage sets. It supports multiple types of window, each of which provides a different method of viewing the scene. A new type of window was created which draws an OpenGL preview of the set, and mirrors it on both the desktop display and touch screen. An example of the touch screen display is shown in figure 2. Touch input can manipulate the viewpoint — single touch drags rotate the viewing angle, while double touches walk the user through the screen. Triple taps reset the viewport.

The target user base is technically naive and as such great effort has been taken to ensure the system is as accessible as possible. Touch is a form of input which users adapt to naturally without training, and users were able to navigate the 3D visualisation of the set efficiently after only a few seconds of familiarisation. By contrast basic mouse interaction skills took far longer to acquire.

By placing the main application on the desktop and supporting multiple ScreenPad clients, an interesting multi-user configuration is created without the complexity of a conventional distributed system. The desktop user retains control, but other users can interact with the program semi-independantly. In the case of StageBuilder, it allowed others working on the show to examine the scenographers work during collaborative review meetings, without needing to learn the main program or crowd round a single screen. Copyright, security and confidentiality issues are also minimised, as the scene file is never transferred to the mobile device (though they can “photograph” the set using their devices standard screen capture facility). There are no versioning issues, as the scene is always held centrally, and there is no danger of the reviewers modifying the set.

Another application currently being examined is integrating a similar review feature into motion capture software, so directors and actors can see the captured data without having to interact with the capture software directly (and without having to leave the MoCap performance area). Key to these kinds of applications is that beyond the initial install of ScreenPad, the user simply turns on their device and is automatically connected to the running application. No further downloads, installs or updates are required, leaving the desktop application in full control.

3. Multi-Touch Inverse Kinematics

While the StageBuilder and Webcam applications were enhanced by the inclusion of the ScreenPad framework, the multi-touch IK application was designed to exploit it fully, and could not exist as a standard desktop, or mobile application.

Multi-Touch control of an animated character is considered in [KN10], but their approach is very different. Their interest in puppetry leads then to eschew direct manipulation in favour of a two handed control system. While this approach works well for their application, it is unsuitable for ours as it requires user investment in both hardware and training.

3.1. Cyclic Coordinate Decent

A standard approach to implementing IK is through a cyclic coordinate decent (CCD) solver [Wel89]. Figure 3 shows a simple IK problem, which can be solved using this approach: The two bones AB and BC are controlled by two rotational joints with the angles θ_a and θ_b respectively. The end point C must be moved to the target T. While analytical solutions may be possible in simple cases, an iterative approach will rapidly converge on a solution if there is one available. Starting with end bone, θ_b is adjusted, rotating C to C' and minimising the distance between C' and T. The result is that BC' has the same orientation as BT. However the problem cannot be solved by simply rotating BC, and an error remains. We therefore progress up the chain of bones, and adjust θ_a to minimise the distance between C'' and T. Again this is achieved by making AC'' have the same orientation as AT. Each step is guaranteed to move C closer to T, and by applying this iteratively a solution is usually found quickly.

Because CCD deals with one joint at a time, it can be easily adapted to incorporate additional constraints such as the rotational limits of joints. Multiple touches by the user create additional targets, which are also simply incorporated. For each target, its effect is propagated once from the touched bone end, back to the root node. This will move the skeleton into a position more favourable to that target, but a single decent will rarely reach the target exactly. The system is then partially solved for the next target. Once all targets have received one decent pass, the whole process is repeated until the system converges on a solution.

In all but the simplest cases IK chains have either multiple solutions or no complete solution. In order to find more subjectively pleasant solutions, damping is applied to that on each iteration a joint is only rotated half way to the theoretically correct angle. This has the effect of finding solutions which move the parent limbs slightly even when a solution exists which moves only the smaller limbs, resulting in a more natural motion. This damping also makes the system more stable. When two targets are in conflict, such that no solution can satisfy both constraints, damping results in an appropriate compromise pose being found, without oscillation.

The system was implemented using ScreenPad, such that

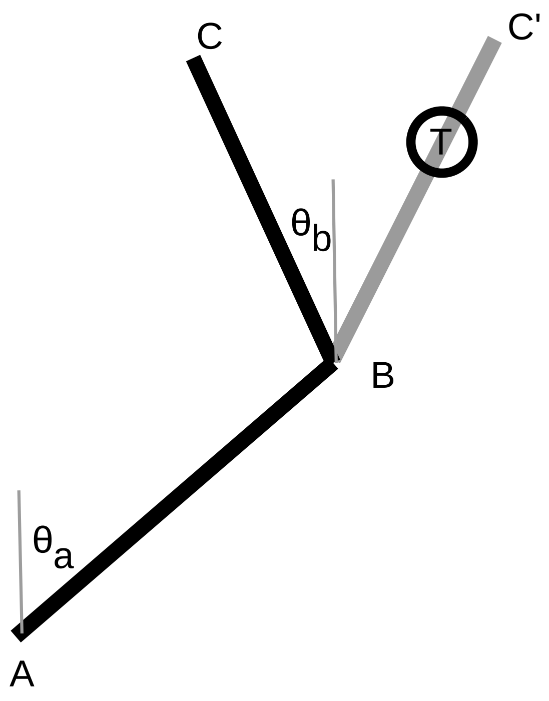

Figure 3: *Cyclic Coordinate Decent.*

touches from the user are sent back to the desktop machine, which binds them to a bone end, and as the touch moves, the CCD solver attempts to move all selected bone ends to their respective targets. The number of touches is limited only by the dexterity of the user. The resultant skeleton is rendered in OpenGL, and the image sent to back to the touch device. For the purposes of illustration a simple skinning algorithm was used to attach a basic model to the skeleton.

3.2. Results

Though the system exhibited quirky behaviour typical of IK implementations, multi-touch provided a quick and intuitive interface for controlling this. All users immediately likened the experience to puppeteering rather than animation, as they had direct real time control over the character, rather than controlling it through an abstract interface.

In addition to the benefits of direct single touch manipulation two forms of multi-touch emerged:

- holding an end effector stationary while moving another joint,
- moving an end effector while holding an intermediate joint.

The first of these is shown in figure 4. The characters left hand is held in the air while the right hand and head are moved. This is equivalent to the technique of pinning, where an end effector is locked to a stationary point (for example an door handle the character is touching) while the rest of the body is positioned. However no special mechanisms are required to implement this within a multitouch system.

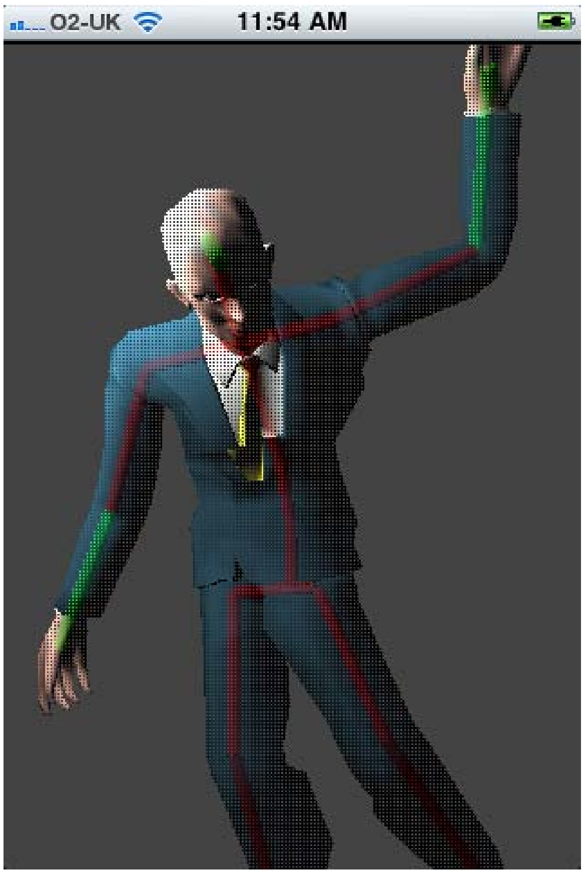

Figure 4: *Multi Touch IK Simulating Pinning.*

The second multi-touch interaction is effectively a form of FK, as shown in figure 5. The characters elbow is held stationary, and the orientation of the forearm can be set explicitly be dragging the hand. This mechanism is not as precise as true FK, as in certain circumstances the hand movement may propagate through the stationary elbow and affect the shoulder, but it is totally intuitive, does not require users to understand the abstract concepts of IK and FK, and avoids having to switch modaly between the two forms of control.

Despite these advantages, multi-touch IK suffers from a lack of accuracy. Touch interfaces are intuitive and quick to use, but fingertips are large and clumsy compared to the accuracy of a traditional pointing device. As such multi-touch IK proved well suited to the rapid posing of characters by users who are not traditional animators, but proved limiting compared conventional approaches to character anima-

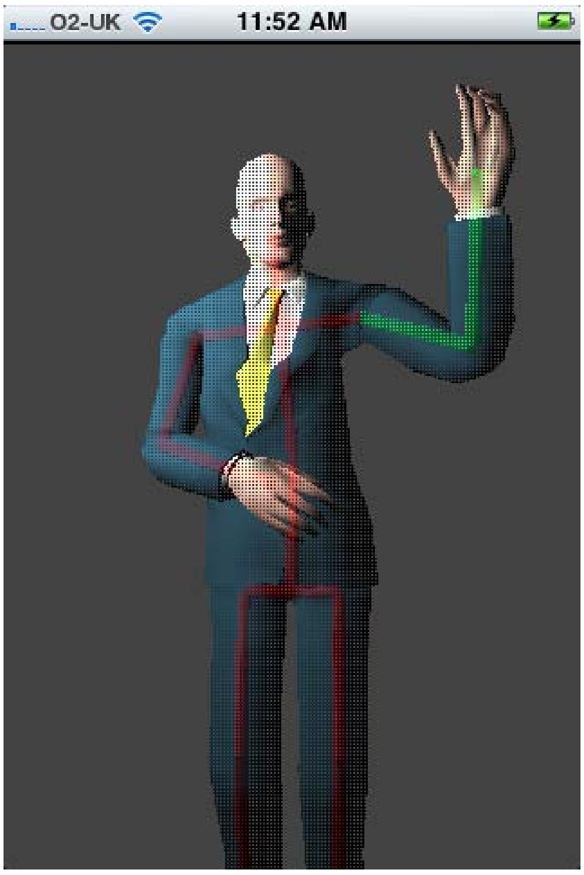

Figure 5: *Multi Touch IK Simulating FK.*

tion where highly trained users can select from a range of tools to generate precise results.

3.3. Direct Multi-touch Vs Gesture

Though iPhone and iPod devices are physically capable of tracking multiple touches and run the ScreenPad software well for other applications, they have particular limitations when used for the multitouch IK. While in principle the iPad is simply a large iPod Touch, its increased screen size creates a qualitatively different user experience.

The smaller devices perform well for single touch direct manipulation, and multi-touch indirect input (such as scaling an image using the pinch gesture), but the screens are simply too small for the average user to accurately touch and then drag multiple points on the screen at the same time, without obscuring the display completely. While still somewhat limited by visibility and the users physical dexterity, the larger display of the iPad allowed the user to select multiple limbs more accurately and position them while still viewing the screen. These levels of touch interaction are summarised in figure 6

	Single Touch	Multi-Touch
Indirect	Simple Trackpad	MultiTouch Trackpad iPhone
Direct	iPhone	iPad

Figure 6: *Forms of touch interaction*

4. Conclusion

The ScreenPad thin client proved a particularly effective approach to implementing multi-touch IK. Interacting with a desktop application though a handheld multi-touch screen was found to be a powerful and versatile extension to the standard interface which was immediately attractive to users. Because ScreenPad is a thin client it can be reused for any number of projects, without requiring users to install additional software on their touch device.

Multi-touch IK is an intuitive technique, which can easily be incorporated into existing animation tools. Only minimal changes were required to a standard CCD solver to support multitouch, and the extra control limited the need for additional features such as pinning or IK/FK switching.

There was a lack of precision, inherent in most touch systems, which would lead experienced animators to prefer existing IK/FK methods, but for casual users the multi-touch interface allows them to quickly pose characters without having to learn either theoretical concepts, or a complex UI. As such it meets our requirements, allowing poseable characters to be incorporated into applications where the characters pose is only a minor part of the users goal.

References

[ALM92] AHLSTRÖM B., LENMAN S., MARMOLIN T.: Overcoming touchscreen user fatigue by workplace design. In *Posters and short talks of the 1992 SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 1992), CHI '92, ACM, pp. 101–102. 2

[App11] APPLE INC: *iOS Developer Program License Agreement*, 2011. 2

[HO99] HAGIWARA S., OLIVER I.: Sega dreamcast: Creating a unified entertainment world. *IEEE Micro 19* (November 1999), 29–35. 2

[HSH*05] HOTELLING S., STRCKON J., HUPPI B., CHAUDHRI I., CHRISTIE G., ORDING B., KERR D. R., IVE J.: Gestures for touch sensitive input devices. United States Patent Application, January 2005. 2

[Kar02] KARWAS P.: Lord of the rings: animation that was not there. In *ACM SIGGRAPH 2002 conference abstracts and applications* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 210–210. 1

[KN10] KIPP M., NGUYEN Q.: Multitouch puppetry: Creating coordinated 3d motion for an articulated arm. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces* (2010), ACM Press. 4

[Nin11] NINTENDO: *Nintendo's Upcoming Wii U*

Console Features Controller With 6.2inch Screen. http://press.nintendo.com/articles.jsp?id=29261, 2011. 2

[SC05] STEINBERG D. H., CHESHIRE S.: *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, December 2005. 3

[SP10] STEPHENSON I., PRIDE R.: Computer Modelling of Theatrical Sets. In *Theory and Practise in Computer Graphics* (Sheffield, United Kingdom, 2010), Collomosse J., Grimstead I., (Eds.), Eurographics Association, pp. 75–81. 4

[Wel89] WELMAN C.: *Inverse Kinematics and Geometric Contrains for Articulated Figure Manipulation*. Master's thesis, Simon Fraser University, 1989. 1, 4