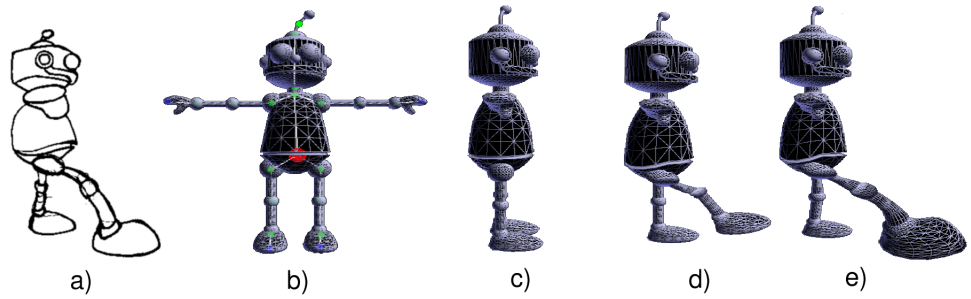
EUROGRAPHICS 2004 / M.-P. Cani and M. Slater
(Guest Editors)

Volume 23 (2004), Number 3

A System for View-Dependent Animation

Parag Chaudhuri†, Prem Kalra† and Subhashis Banerjee†

Department of Computer Science and Engineering,
Indian Institute of Technology Delhi, New Delhi, India.



Our system takes as input a sketch (a), and a base mesh model (b), then recovers a camera to orient the base mesh (c), then reconstructs the skeleton pose (d), and finally deforms the mesh to find the best possible match with the sketch (e).

Abstract

In this paper, we present a novel system for facilitating the creation of stylized view-dependent 3D animation. Our system harnesses the skill and intuition of a traditionally trained animator by providing a convivial sketch based 2D to 3D interface. A base mesh model of the character can be modified to match closely to an input sketch, with minimal user interaction. To do this, we recover the best camera from the intended view direction in the sketch using robust computer vision techniques. This aligns the mesh model with the sketch. We then deform the 3D character in two stages - first we reconstruct the best matching skeletal pose from the sketch and then we deform the mesh geometry. We introduce techniques to incorporate deformations in the view-dependent setting. This allows us to set up view-dependent models for animation.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Animation

1. Introduction

Traditionally animated 2D characters are remarkably expressive. To enhance the artistic impact of the characters and to impart them a personality of their own, animators often stylize the appearance of these characters. The stylization may vary the appearance of such characters with time, viewing direction, their mood or their pose.

Such stylizations have also been applied to 3D animated characters. Applying the 3D analogue of the traditional 2D methods requires considerable skill and hard work on the part of the animator. For example, it may be relatively simpler to change the appearance of a character in 2D by drawing it in another pose, but posing a character in 3D is difficult. The number of aspects or parameters which have to be controlled/manipulated is overwhelming. One may attempt to use existing animation and modeling tools to modify the 3D object at selected keyframes to match the 2D sketches. This approach is tedious and might be prohibitively expensive.

† Email: {parag, pkalra, suban}@cse.iitd.ernet.in

Our goal is to build a system to interface the rich stylization of traditional 2D animation with 3D character animation. The system aims to lessen the burden on the animator while transferring a 2D character animation to 3D. We present a system which harnesses the skill and intuition of the traditionally trained animator by providing a sketch based input mechanism.

We provide a technique of injecting view-dependent stylization into the animation. View-dependent stylization allows the animator to make the character respond automatically to changes in the view direction. The usual technique of view-dependent geometry [Rad99] allows for such stylization, however, here the creation of view-dependent models is quite cumbersome. We introduce a system which provides a more intuitive and better way of performing view-dependent deformations.

We begin by providing the background for our work, examining and comparing related techniques in Section 2. Then we present a short overview of our system in Section 3. We give details of the *View-Dependent Deformation System* in Section 4 and the *View-Dependent Animation System* in Section 5. Finally, we present and discuss the results we have produced using our system in Section 6. We conclude with Section 7.

2. Background

In recent years many of the techniques of traditional 2D animation which give it a unique expressiveness and aesthetic aspect have been interfaced to 3D animation systems. To this effect several artwork styles have been explored in non-photorealistic rendering literature. Many of these algorithms focus on a specific artistic style, or a closely related family of styles.

Efforts have also been made to capture the expressive movement styles usually found in traditional animation. [BLCD02] describe a method to capture the motion from a cartoon animation. They use a combination of affine transformations and key-shape interpolation to track 2D contour deformations. This can then be re-targeted to either another 2D or 3D character. However, they require the corresponding keyframes of the target character to be manually constructed. Though we address a different kind of stylized animation in this paper, our method, in part, focuses on assisting the creation of these keyframes while they provide a method to interpolate between them.

Interfacing 2D-3D animations is generally a very tedious task. A very useful and common approach for this purpose is to provide a sketch based interface. In fact, the standard way of documenting a character is to show the characteristic poses of the character in a *model sheet* [TJ84]. This concept inspires our work to permit animators to stylize animations by specifying the appearance of the characters in characteristic poses in the form of sketches. The appearance of a character depends on the character's form and the viewing direction. In the real world appearance is primarily based on the underlying 3D shape. Our system offers a mechanism to specify a correspondence between the sketched characteristic pose in 2D and the inherent 3D structure of the character.

Early attempts at posing stick skeleton figures from sketches were made by [Sab91], where the system used a simple reverse projection algorithm to reconstruct a skeleton pose from the sketch. The projection considered in this system is orthographic and it resolves the depth ambiguity by user interaction. In another work [HP92] developed a sketch based animation system using a touch sensitive tablet. However their system relies completely on the artist to resolve ambiguity. In a recent work, [DAC*03] also provide a simple sketching interface for articulated figure animation. The user draws the skeleton on top of the 2D sketch. Then they reconstruct the various alternative poses of a 3D stick figure corresponding to the 2D pose. The user is allowed to pick the desired pose and perform a few corrections to it. The skeleton posing technique used here is in spirit similar to the one used in our system. However, our system goes significantly beyond theirs, as they assume a scaled orthographic camera for reconstructing the 3D pose whereas we can recover the best full projective camera and view direction which matches the sketch. We show that this flexibility in the recovery of the camera allows for dramatic effects, such as close-up shots, in the resulting animation. This camera recovery needs only minimal user interaction in the form of a few mouse clicks to specify correspondences. Our system also allows the user to correct/refine the automatically reconstructed poses. We further match the deformation of the character's mesh to the sketch which cannot just be recovered by matching the skeletal pose.

In another contemporary work, [LGXS03] present a method for stylizing animations through drawings. They allow a traditional animator to modify frames in the rendered animation by redrawing the key features such as silhouette curves. These changes are then integrated into the animation. To perform this integration, they divide the changes into those that can be made by altering the skeletal animation, and those that must be made by altering the character's mesh geometry. To propagate mesh changes into other frames, they use a modified image warping technique which takes into account the character's structure. We differ from the approach presented in this paper, where the skeletal deformations are obtained by manually modifying a standard motion capture stream and not automatically recovered from a sketch. This paper has no notion of a camera recovery. It infact relies on the animator to find the frame in the original rendered animation that best matches the sketch and then creates a deformation field to warp the silhouette by matching the curves in 2D. Our system addresses a much wider problem and is more flexible.

View-Dependent geometry (VDG) as introduced

by [Rad99] also provides a sketch based input mechanism. It is a method for specifying models in such a manner that their *geometry can change with the view point*. Our definition of VDG is the same as given in this paper. View specific distortions form a very elegant method for specifying stylized animations. The inputs to the system are a 3D model of a character (the base model) and a set of drawings of the character from various viewpoints. The user then manually matches the viewing direction and the shape of the base model with each drawing thus creating a *key viewpoint* and *key deformation* pair for every drawing. A key viewpoint and a key deformation pair together constitute a *view-dependent model*. All the view-dependent models together form a *view-sphere*. Tracing any camera path on this view-sphere generates the appropriate animation with view-dependent deformations. The method also supports creation of animated view-dependent models, where the base model is non-rigidly animated and a single set of key deformations is no longer sufficient. Such situations need a different set of key deformations for the keyframes of the model's animation. This essentially results in a separate view-sphere at each keyframe. the animation is generated by blending the deformations on a per-frame basis in response to the current viewpoint as the viewpoint moves from one view sphere to another.

One of the major drawbacks of this technique is the amount of manual intervention required by the user in the creation of the view-dependent models. [KKB02] proposed a method to automate some aspects of the view-dependent model creation. However, the view alignment and the deformation modeling techniques used are very rudimentary and non-intuitive. Our system allows for automated recovery of the *key viewpoint* and creation of view-dependent models. We extend the technique to allow for skeletal deformation, and apply the mesh deformations relative to the underlying skeletal deformation. This allows us to blend together normal (i.e. non view-dependent) animation with view-dependent animation.

Contributions Our system allows the recovery of a camera which best matches the intended view direction in the sketch. We go beyond existing systems as we can recover the full projective camera which allows more expressive animation. We use robust techniques from computer vision to do so. Our system can reconstruct the skeletal pose of the 3D character, which is also consistent with the recovered camera. The system can also deform the mesh to match the deformation of the sketched character. This is done relative to the skeletal pose already recovered. We introduce a method of doing view-dependent deformations. The deformation model is coupled with the camera, and thus it gives deformation consistent with the recovered view point. At every stage the system offers considerable flexibility to the user to correct the recovered view, reconstructed pose or the deformed mesh by manual intervention.

3. System overview

Our system is organized as follows. The system has two major components. The first is a *view-dependent deformation (VDD) system*. The VDD system requires the following inputs: the sketches of the character in various poses and a 3D mesh model of the character which we call the *base mesh model*. We want to pose our base mesh model according to the sketched poses and recover the intended view directions for each sketch. The VDD system modifies the base mesh to match the sketched pose.

We embed a skeleton inside the mesh model using a simple interactive tool which we have developed. We next construct a lattice made up of tetrahedral cells, enclosing the mesh, using an interactive tool developed by us. We recover a camera which best aligns our base mesh with the viewing direction in the sketch. For this the user needs to specify joint correspondences between the embedded skeleton and the sketched pose. Using these correspondences, we set up a system of equations and solve robustly for the best possible camera which matches the viewing direction in the 2D sketch of the character. We use this recovered camera to drive a view-dependent deformation model. Our deformation model has a multi-layered structure. At the first stage it uses an inverse kinematics (IK) layer. This matches the pose of the embedded skeleton to the sketched pose. Inverse kinematics by itself though very powerful, is quite unwieldy and difficult to use to precisely pose a character in 3D. We have integrated the IK layer in the view-dependent framework which constrains it to find poses very close to the desired pose. At the second stage it uses a lattice based direct free form deformation (DFFD) layer. We have also integrated this mesh deformation model with the recovered view, so the mesh can be deformed in accordance with the recovered view and a best possible match with the sketched character is obtained. This multi-layered deformation technique allows us to cover a wide range of deformation styles. This view recovery and deformation process is repeated for each of the sketched poses. At the end of this stage we have a set of recovered views and corresponding deformed meshes - these are the *key viewpoint* and *key deformation* pairs (introduced in the previous section) respectively.

This set of *view-dependent models* is passed on to the second major component, the *view-dependent animation (VDA) system*. The animation system actually generates at run time the *view-sphere* and all the frames in response to the camera path traced on this sphere. The VDA system further has the capability to blend together the normal (i.e. non-view dependent) and the view-dependent segments of the animation.

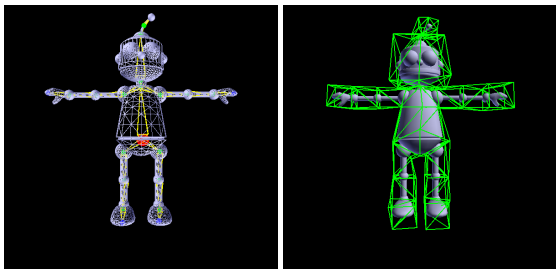
In the following sections we present our techniques in detail. We also look at some of the implementation issues involved in building our system. We first describe the VDD system in the following section.

4. The View-Dependent Deformation (VDD) System

We describe here how the VDD system is used to recover a camera from a given sketch and how it is subsequently used to deform the *base mesh* model to match the sketch. We use the term *posed mesh* model to describe the state of the mesh model after the best matching skeletal pose has been determined, and the term *deformed mesh* model to describe the state of the mesh model after it has been deformed using DFFD to best match the sketched character.

4.1. The inputs

The primary inputs to the system are a sketched pose provided by the animator and a base mesh model. The base mesh model is a triangulated mesh of the character to be animated, which can be made using any modeling software. The system also requires two secondary inputs - a skeleton and a lattice. We have implemented simple interactive techniques for skeleton and lattice construction. We give an example of a skeleton embedded in the mesh using this technique in Figure 1(a). The lattice is made up of tetrahedral cells which enclose the mesh. The lattice is defined in a manner such that each lattice cell is uniquely associated with some skeleton bone. Thus the lattice construction is based on the way the underlying skeleton has been defined. We give an example of such a lattice in Figure 1(b).



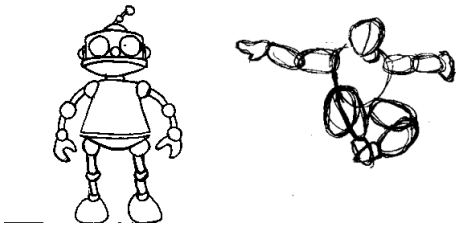
(a) (b)

Figure 1: *An example of a skeleton embedded inside a mesh model, and a lattice enclosing the mesh model*

4.2. Camera Recovery Engine

We first recover the view direction and the camera which best matches the sketch. The user clicks correspondences between the sketched pose and the skeleton joints. The skeleton joints are marked on the sketched pose.

Our system can handle two broad categories of sketches. The sketch may actually be a sketch of the character in question. For such sketches, correspondences are very easy to specify as the user can locate the joints on the sketch easily and accurately (see Figure 2(a)). The second category of sketches that the system can handle are rough *mannequin* sketches (such sketches are easier and faster to draw and are often used for rough pose planning during animation). If the bone proportions of the mannequin are roughly the same as that of the character then locating the skeleton joints on such a sketch is quite intuitive for the animator and correspondences can be approximately specified very quickly (see Figure 2(b)). The camera recovery engine is robust enough to determine a feasible camera for the approximately placed joints and so works equally well with both these categories of sketches. We demonstrate the use of mannequin sketches through one of our results (see Figure 6(b)), where the whole animation has been generated from such sketches. Note that for the mannequin sketches (or for that matter any sort of approximate sketches) to work, their overall scale and limb proportions should be similar to the model.

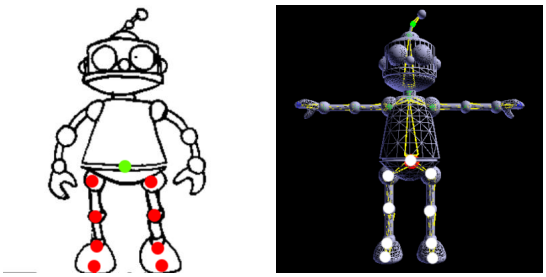


(a) A sketch of a character (b) A sketch of a mannequin

Figure 2: *Possible input sketch types*

The joints used during camera recovery have to be rigid relative to change in pose i.e., the joints which have not moved from the base mesh model to the posed mesh model. So the user needs to click the position of these rigid joints (see Figures 3(a) and 3(b)). The first point marked on the image must correspond to the root of the skeleton. This allows the system to reposition the image coordinate system origin accordingly (as the root forms the origin of the skeleton's coordinate system). The minimum number of joints whose positions must be clicked on the sketch can vary from 3 to 6 depending on the type of the camera to be recovered. If sufficient number of rigid joints cannot be found on the skeleton, the user can locate any point on the sketch which has not moved relative to the base mesh model. The corresponding 3D points to these points may be a point on the skeleton (an artificial joint) or a point on the base mesh itself.

After this the two point lists (2D sketch points and 3D skeleton joints) are normalized. The full projective camera is computed using the *Normalized Direct Linear Transformation Algorithm* and the affine camera is computed using the *Gold Standard Algorithm* as given in [HZ00]. These are numerically robust techniques, and work well with hand



(a) A sketch with rigid joints marked

(b) The base mesh model with the skeleton

Figure 3: *Marking of joint correspondences*

clicked correspondences (for a detailed description of the numerical behaviour of these techniques see [HZ00]). The weak perspective and orthographic cameras are recovered using techniques similar to those used for affine cameras. We have most commonly worked with the full projective and affine cameras. The full projective camera is better suited for cases where close-ups of the characters are required (i.e. the distance of the camera from the character is comparable to the width of the character along the look-at direction). The affine and other cameras are better for cases where the camera is further away from the character (i.e. the distance of the camera from the character is considerably more than the width of the character along the look-at direction) and hence the perspective foreshortening effect is not pronounced. The full projective camera is of the form:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \quad (1)$$

The camera thus estimated will project the clicked joints on the corresponding points on the sketch. Now if the camera recovered be P and the camera centre be C then we must have:

$$P \cdot C = 0 \quad (2)$$

So the camera centre is recovered as the right null space of the camera matrix. For the full projective camera we get a finite camera centre, but in case of the affine, weak perspective and orthographic cameras the camera centre is a point at infinity. We, however, are only interested in the camera view direction, i.e., the unit vector in the direction of the line joining the camera centre with the root of the skeleton and this is recoverable in all cases. We assume here that we are looking towards the root of the skeleton. When we look towards the skeleton using the recovered camera the mesh model appears aligned with the sketched pose. We refer to this recovered viewpoint as a *key viewpoint*.

4.3. Deformation Engine

Now the mesh model has to be deformed to find its best possible match with the sketched pose. This is done by the deformation engine of the VDD system. We use a two layered deformation engine. We look at both the layers one by one.

4.3.1. The inverse kinematics layer

The IK layer is used to pose the skeleton. We use an exponential map parametrization for joint rotations as given by [Gra98]. We compute the Jacobian using the Singular Value Decomposition. Once the Jacobian is obtained, we can solve for the change in the joint state vector. Here we find a least squares solution using the pseudo-inverse of the Jacobian. However, near a singularity, the problem becomes ill-conditioned and the norm of resulting least squares solution may tend to infinity. So it needs to be regularized which we have done using damped least squares. Once the pose of a chain has been computed the state vector for the chain is updated appropriately and the change in joint transformations is propagated throughout the skeleton to pose the skeleton. A detailed description of the techniques mentioned above may be found in [Wel93] and [Bae01].

The IK layer poses the skeleton. For obtaining the posed mesh model, the mesh is made to move with the skeleton using in essence a *blended skinning* approach. We use the lattice which encloses the mesh to do this. During construction, every cell of the lattice is uniquely associated to some skeleton bone. The cell vertices may be shared among two or more cells. So the cell vertices have weights assigned to them which denote the degree of influence of the corresponding underlying skeleton bones on those vertices. Every lattice cell vertex also has its own coordinates recomputed in the local coordinate system of their associated bones. So when the underlying bones move, the new position of the lattice cell vertex is computed using *bones blending* [KZ03]. Every lattice cell is also associated with all the mesh vertices contained inside the tetrahedral lattice cell. Every mesh vertex is associated to atmost one lattice cell. To move the mesh, the system calculates the new position of the mesh points contained in a lattice cell using their barycentric coordinates. This results in a smooth deformation of the mesh as the skeleton is moved. The IK layer poses the skeleton using a *view-dependent posing algorithm*.

The view-dependent posing algorithm

Now we present the algorithm for automatically posing the base model of the character. The user first needs to specify the joint correspondences on the sketch for all the joints which need to be moved during the posing. These were not marked during the camera recovery phase (see Section 4.2).

We define a few terms which we use in our posing algorithm. The skeleton forms a rooted tree. We define a simple kinematic chain as any sequence of joints in this skeleton

such that the sequence does not span across branches in the tree. We define the starting joint of this kinematic chain as the root joint for the chain and the ending joint as the end-effector. Note that the end-effector can even be a joint which is not a leaf, as the chain can start and end anywhere as long as it does not span a branch. The goal is the desired 3D position of the end-effector of the kinematic chain. The algorithm now proceeds as given in Algorithm 1. The user marks

Algorithm 1: *View-Dependent Posing Algorithm*

Require: The camera must have been estimated before this algorithm can be run.
Require: Correspondences for all the joints to be deformed must be marked.
1: **repeat**
2: Select a simple kinematic chain
3: Back project 2D end-effector position as the 3D goal using the recovered camera
4: Run IK to make the chain reach for the goal
5: **until** the desired pose has been achieved

out a kinematic chain which needs to be deformed. The user only has to click on the joint names in a hierarchical GUI to identify the joints in the chain on the sketch and on the skeleton. The position of the end-effector is back projected from the sketch into 3D space using the pseudo-inverse of the recovered camera (see Figure 4). This is done in the following manner. Let the end-effector position on the sketch

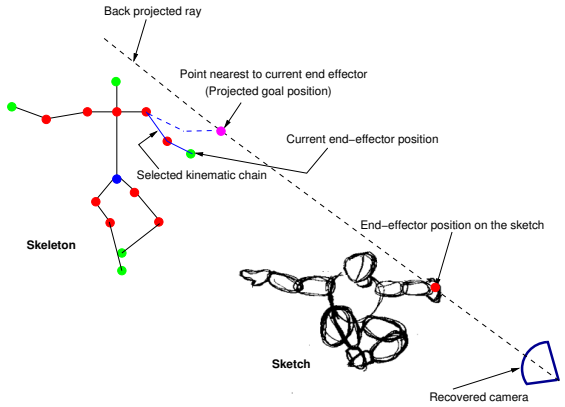


Figure 4: *Projecting the goal back using the recovered camera*

be $e \equiv (x_e, y_e)$ and the recovered camera be P (P is of the form given in Equation 1) with the camera centre as C then projecting e back using P^{-1} (the pseudo-inverse of P) gives us a 3D ray R_e

$$R_e = C + \lambda \cdot (e \cdot P^{-1}) \tag{3}$$

Now we find the point (X,Y,Z) on this ray which is closest to the current position of the end-effector joint in 3D. If the current end-effector position is given by $E \equiv (X_e, Y_e, Z_e)$ then the new position of the end-effector E_{new} is given by the solution of the following minimization:

$$\min_{(X,Y,Z)} = (X - X_e)^2 + (Y - Y_e)^2 + (Z - Z_e)^2$$

subject to:

$$\begin{aligned}(p_{11} - p_{31}x_e)\cdot X + (p_{12} - p_{32}x_e)\cdot Y + \\ + (p_{13} - p_{33}x_e)\cdot Z + p_{14} &= p_{34}\cdot x_e \\ (p_{21} - p_{31}y_e)\cdot X + (p_{22} - p_{32}y_e)\cdot Y + \\ + (p_{23} - p_{33}y_e)\cdot Z + p_{24} &= p_{34}\cdot y_e \end{aligned} \tag{4}$$

E_{new} is the goal position in 3D. The constraints arise from the fact that P must project the point (X,Y,Z) to $e \equiv (x_e, y_e)$ on the sketch. So the constraints embody the camera projection equation. Once we have projected the goal, the IK layer takes over and deforms the chain in 3D to make it reach for the goal (see Figure 4). So the IK layer is tied to the camera recovered and is guided by it to find a pose which satisfy the view-dependent constraint, so the projection of the posed mesh model matches (in terms of the skeletal pose) the sketch when projected using the recovered camera.

IK finds the best possible configuration of the chain to make it reach the goal. However, this may not be the desired configuration for the chain. Often the easiest way around this is to consider short chains (i.e. chains with upto 3 segments). In all the experiments we performed the IK layer always found the correct chain configuration for short chains. Selecting short chains mean more number of chains have to be selected but even then the number of joints to be clicked are about 2 - 10. This is still much easier than posing directly in 3D. The IK layer always selects the point on the ray which is closest to the current end-effector position as the new goal. This is a very valid assumption as the new goal position is being selected as a point causing least movement/change from the existing end-effector position. If the above process does not pose the chain to the users satisfaction then the user can also correct the pose of the chain by interactively tweaking the bone positions through the IK layer.

This process is repeated for every chain till the desired pose of the complete skeleton is achieved. In the current implementation the posing process poses a single chain at a time. This can be easily extended to simultaneous solution of multiple chains reaching for multiple goals using techniques given in [BMW87]. The lattice deforms the mesh whenever IK repositions a chain. Thus at the end of the posing phase we have a the *posed mesh model*.

Further, we can pose elements of the character which may not be there in the sketch using the interactive posing facility available in the IK layer. For example, in one of our results the character has a tail but the input sketches are mannequin sketches (see Figure 6(b)) and do not have a tail in them, so the tail is interactively posed by the user in this stage.

If the input sketches are those of the mannequin type then

the user can proceed only upto this stage and must move on to the VDA system from here. In such a case the posed mesh models become the *key deformations* associated with the recovered camera's which are the *key viewpoints* and so constitute valid *view-dependent models*. It may be argued that these are just posed and not deformed but the geometry of the mesh model has changed from the base mesh model (due to the lattice layer moving the mesh along with the skeleton) and animating using the VDA system produces a view-dependent animation where the geometry responds to changes in view. We illustrate this with an animation sequence in our results (see Section 6).

However, if the animator has sketched the character itself then the mesh is deformed using a view-dependent mesh deformation algorithm to make it match with the sketched character and get a better *view-dependent model*. This forms the second layer of our deformation engine.

4.3.2. The direct free-form deformation layer

The mesh is deformed in the second layer of the deformation engine using the technique of direct free-form deformation (DFFD) [HHK92] [KO03]. The lattice is made of tetrahedron cells. The tetrahedron is a simplex and hence allows a linear barycentric basis. Using this basis to parameterize the mesh points contained in the lattice allows us to use the lattice for DFFD. The system allows the user to interactively deform the mesh points by direct manipulation with mouse clicks. The user can move groups of points as well as individual mesh points. For automatic view consistent deformation of the mesh we have a view-dependent mesh deformation algorithm which couples the process of deforming the mesh with the recovered camera. We describe this algorithm below.

The view-dependent mesh deformation algorithm

The user first needs to mark correspondences by clicking points on the sketch and the posed mesh model. These points are the inputs to the view-dependent mesh deformation algorithm. We call these points the input mesh points. These points usually belong to the silhouette curve of the sketch as we want to match the silhouette of the sketch with the mesh. The points in 3D where these input points are to be moved are called the target points. The algorithm works as given in Algorithm 2. The back projection and target point selection is done using exactly the same manner as described in Section 4.3.1 for skeleton posing. In this case $e \equiv (x_e, y_e)$ become the 2D points on the sketch. P is again the recovered camera with the camera centre as C. If input 3D point is $E \equiv (X_e, Y_e, Z_e)$ the target point (X, Y, Z) can be found by solving Equation 4 (see Figure 5). The constraints ensure that P projects the computed target point (X, Y, Z) to e on the sketch. When the input mesh points move to the target points, the DFFD algorithm recalculates the position of the affected lattice cell vertices keeping the basis parameterization of the input mesh points same. The rest of mesh vertices

Algorithm 2: *View-Dependent Mesh Deformation Algorithm*

Require: The camera must have been estimated before this algorithm can be run.
Require: Correspondences for all the input points to be moved must be marked.
1: **repeat**
2: The 2D points on the sketch are projected back in 3D space as a ray, using the recovered camera
3: The points on these rays which are closest to their corresponding points in 3D are chosen as the target points
4: The set of input points and target points are passed on to the DFFD layer which actually moves the points.
5: **until** the desired deformation is achieved

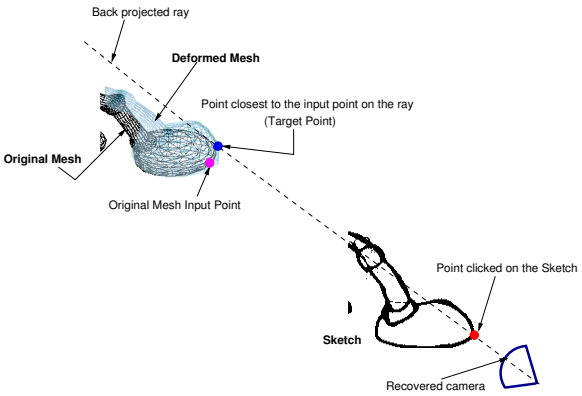


Figure 5: *Projecting the input mesh point back using the recovered camera and deforming the mesh*

in the affected lattice cells suitably deform when the lattice cell vertices move. If the above algorithm does not deform the mesh to the user's satisfaction, the user can interactively refine the mesh deformation by manually adjusting the mesh points.

Note that if the lattice is considerably coarse as compared to the mesh, its effect is not localized to a very small area. The system has also the facility to use a radial decay function (this feature can be turned on/off by the the user) which localizes the effect of the moved mesh vertices to smaller regions around them rather than to all the vertices inside the affected lattice cell. In such a case the movement of a mesh vertex is damped by a weighting factor calculated using a decay function. A vertex may get assigned multiple weights when more than one mesh point is moved. In such a case only the maximum of all those weights is finally retained. So if a mesh point lies outside the radial region of influence of the input mesh point it does not move at all. The barycentric parameterization of the mesh vertices changes

when the radial decay function is used and has to be recomputed every time, but because it is a linear parameterization, recalculation is not costly. Since the mesh deformation is being applied on the posed mesh model and hence is relative to skeleton pose computed, this scheme works quite well in most cases. The layering of the deformation engine allows the user to look at and manipulate suitably the skeletal and non-skeletal (or elastic) deformation separately without causing unwanted artifacts in the mesh (as is also observed in [LGXS03] who also match the silhouette but use curve matching in 2D). Though an adaptive hierarchical lattice will be a better solution for the localized deformation problem than the radial decay function, but the decay function is much easier to implement. The mesh obtained after this stage is the *deformed mesh model*.

This process of camera recovery, posing and mesh deformation is repeated for each of the character sketches and we get a set of *view-dependent models*. These are passed on to the VDA system for animation. Next, we describe the VDA system.

5. The View-Dependent Animation System

The VDA system will generate the view-dependent animation. To create a view-dependent animation sequence, we normalize all the recovered key viewpoints to a unit sphere which surrounds the base model. This is called the *view sphere*. Then we partition the space inside the view sphere by computing its 3D convex hull. Now we want to get the deformed mesh model for any new viewpoint. To get the desired deformation the system first finds the closest three *key viewpoints*. The new viewpoint will lie in the triangle formed by these three key viewpoints and hence will have barycentric coordinates, say given by w_1, w_2, w_3. To get the new deformed mesh at this new viewpoint, every vertex of the mesh (say v) is computed as a barycentric combination of the corresponding vertices in the three *key deformations* or deformed meshes associated with the afore mentioned key viewpoints (say v^i, v^j, v^k) as:

$$v = w_1 \cdot v^i + w_2 \cdot v^j + w_3 \cdot v^k \tag{5}$$

The barycentric interpolant is used here because it is linear and simple to compute. When one of the barycentric weights reaches a value of 1 and the other two weights are equal to zero, the current viewpoint exactly matches a key viewpoint (i.e., when the current viewpoint is at a triangle vertex). When this occurs, the new deformed model will exactly match the one key deformation corresponding to that viewpoint. The barycentric weights can be scaled exponentially to alter the sharpness of the transition between adjacent deformations as done in [Rad99].

Now animating in the view space consists merely tracing the camera path on the sphere and the animation is generated automatically in response to the view. [LGXS03] has to specifically propagate the mesh deformations across keyframes, but in our case since the new deformed mesh is already a blended combination of the key deformations, we do not have to worry about this. For creating view dependent animations where the base model moves non-rigidly in time we use the technique given in [Rad99] for creating the *animated view dependent models* (see Section 2).

We have coupled view-dependent geometry with IK using the embedded skeleton, so we can combine the types of animation these two techniques independently generate. The animator might want to intersperse sequences of normal animation (where the geometry does not respond to the view point) with view-dependent animation. We obtain all the keyframes required to create the normal animation sequences using interactive IK. We then animate these using keyframe interpolation techniques using quaternion *slerp* interpolation [Sho85]. If a view-dependent sequence follows the IK sequence, then we make the last computed IK pose a key-deformation of the view-dependent sequence and recover the camera for it if a sketch is available or assume the camera direction to be the same as in the IK part. This allows a smooth transition from a pure IK sequence to a view-dependent sequence. A similar technique applies if an IK sequence follows a view-dependent sequence. Our results demonstrate this smooth blending of the two animation types. Finally the VDA system renders the animation.

6. Results

We now discuss the animation sequences we have generated using our system. Each of these animation sequences tries to demonstrate the various capabilities of our system.

The *first* animation sequence is a concept demonstration. The character used in the sequence is called Hugo. The sketches used to generate the animation and the corresponding keyframes generated by our system are given in Figure 6(a). We have recovered a full projective camera for this animation.

The *second* animation is inspired by the opening *freeze frame - camera rotate* shot from the movie Matrix [WW99] filmed on the character called Trinity. The character used in our animation is the Olaf the Ogre. As the camera goes round Olaf, he replicates the mid-air kick made famous by the movie. The mannequin sketches and the corresponding rendered keyframes are given in Figure 6(b). The camera model used in this sequence is affine.

In final animation clip titled "Ballet of the Hand," the character used is a cartoon hand (and hence the four fingers). Affine cameras are used in this animation. The animation shows the blending of an pure IK sequence with a view-dependent sequence as it has has three view-dependent sequences interspersed with pure IK sequences. The animation also has a complex set of camera and character movements. We give the some of the sketches and the corresponding rendered keyframes from the animation in Figure 6(c).

7. Conclusions

We have introduced a novel way of doing view-dependent deformations. We have showed how to embed a multi-layered deformation system into a view-dependent setting and integrate it with computer vision techniques for camera estimation. We have enhanced the technique of view-dependent geometry by tying it up with the more conventional 3D character manipulation technique of inverse kinematics and direct free form deformation. The system allows the creation of view-specific distortions as a character moves from the traditional 2D world to the modern day 3D world of computer animation. We have strived to make the process of viewpoint alignment and model deformation semi-automated and more intuitive for the animator. We believe that this will help the animator in better effecting the stylizations he/she wants in the animation. The system while losing none of the original power of view-dependent geometry creates an easier to use and stronger framework for a sketch based interface to 3D animation.

We observe that the system naturally lends itself to further automation. The system can be enhanced to handle multi-modal input, i.e., from sketches, still photographs and video as well. This would lead to a further amalgamation of the real and the virtual worlds, thus producing rich animations.

Acknowledgments

We would like to thank Ashwani Jindal for implementing the interactive skeleton and lattice making tools. Hugo's mesh is courtesy Laurence Boissieux, INRIA.

References

[Bae01] BAERLOCHER P.: *Inverse kinematics techniques for the interactive posture control of articulated figures*. PhD thesis, EPFL, 2001.

[BLCD02] BREGLER C., LOEB L., CHUANG E., DESHPANDE H.: Turning to the masters: Motion capturing cartoons. *ACM Transactions on Graphics 21*, 3 (July 2002), 399–407.

[BMW87] BADLER N. I., MANOOCHEHRI K. H., WALTERS G.: Articulated figure positioning by multiple constraints. *IEEE Computer Graphics & Applications 7*, 6 (June 1987), 28–38.

[DAC*03] DAVIS J., AGRAWALA M., CHUANG E., POPOVIĆ Z., SALESIN D.: A sketching interface for articulated figure animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), pp. 320–328.

[Gra98] GRASSIA F. S.: Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools 3*, 3 (1998), 29–48.

[HHK92] HSU W. M., HUGHES J. F., KAUFMAN H.: Direct manipulation of free-form deformations. In *Computer Graphics (Proceedings of SIGGRAPH 92)* (July 1992), vol. 26, pp. 177–184.

[HP92] HECKER R., PERLIN K.: Controlling 3d objects by sketching 2d views. In *SPIE - Sensor Fusion V* (Nov. 1992), vol. 1828, pp. 46–48.

[HZ00] HARTLEY R. I., ZISSERMAN A.: *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521623049, 2000.

[KKB02] KATE R. J., KALRA P., BANERJEE S.: Towards an automatic approach for view-dependent geometry. *International Journal of Image and Graphics (IJIG) 2*, 3 (2002), 413–423.

[KO03] KOBAYASHI K. G., OOTSUBO K.: t-FFD: free-form deformation by using triangular mesh. In *Proceedings of the eighth ACM symposium on Solid modeling and applications* (2003), ACM Press, pp. 226–234.

[KZ03] KAVAN L., ZÁRA J.: Real time skin deformation with bones blending. *WSCG Short Papers proceedings* (2003).

[LGXS03] LI Y., GLEICHER M., XU Y.-Q., SHUM H.-Y.: Stylizing motion with drawings. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), pp. 309–319.

[Rad99] RADEMACHER P.: View-dependent geometry. In *Siggraph 1999, Computer Graphics Proceedings* (1999), pp. 439–446.

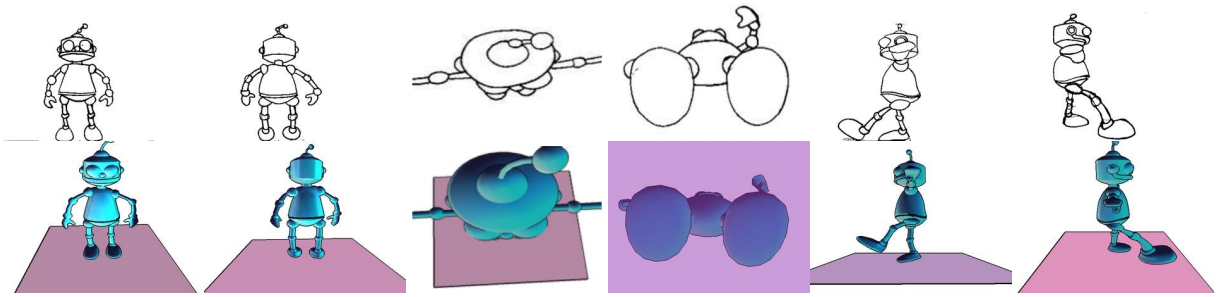
[Sab91] SABISTON W. R.: *Extracting 3D Motion from Hand-Drawn Animated Figures*. Master's thesis, Massachusetts Institute of Technology, Master of Science in Visual Studies, June 1991.

[Sho85] SHOEMAKE K.: Animating rotation with quaternion curves. *Computer Graphics (Proceedings of SIGGRAPH 85) 19*, 3 (July 1985), 245–254.

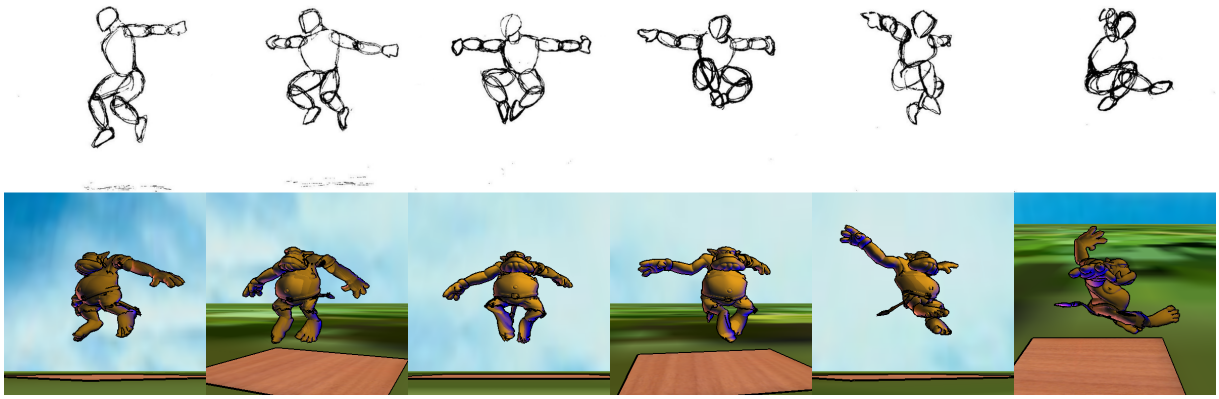
[TJ84] THOMAS F., JOHNSON O.: *Disney Animation: The Illusion of Life*. Abbeville Press, 1984.

[Wel93] WELMAN C.: *Inverse kinematics and geometric constraints for articulated figure manipulation*. Master's thesis, Simon Fraser University, 1993.

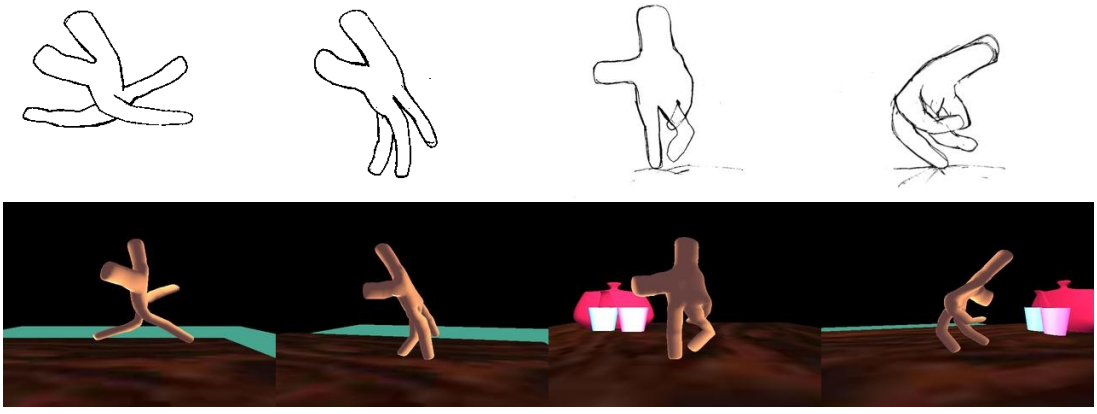
[WW99] WACHOWSKI L., WACHOWSKI A.: The Matrix. *Warner Studios* (1999).



(a) Hugo sketched from six directions namely front, back, top, bottom, left and right. The second row shows the corresponding animation keyframes generated by our system. Notice the marked perspective foreshortening in the top and bottom view sketches and how the effect is correctly reproduced by the recovered camera. In the last sketch Hugo's right leg undergoes a marked mesh deformation which cannot just be obtained by skeletal pose recovery. This is as per the corresponding sketched view.



(b) This sequence has been generated using mannequin sketches. This demonstrates that view-dependent animation is indeed possible using such sketches and the skeletal pose reconstructed by our system. We stress here that all the changes in Olaf's pose is in response to the camera movement and it should not be confused with a rigid rotation of the character.



(c) Sketches and rendered keyframes from the "Ballet of the Hand" animation.

Figure 6: *Results generated using our system.*