View-Dependent Geometry

Paul Rademacher
University of North Carolina at Chapel Hill

ABSTRACT

When constructing 3D geometry for use in cel animation, the reference drawings of the object or character often contain various view-specific distortions, which cannot be captured with conventional 3D models. In this work we present a technique called View-Dependent Geometry, wherein a 3D model changes shape based on the direction it is viewed from. A view-dependent model consists of a *base model*, a set of *key deformations* (deformed versions of the base model), and a set of corresponding *key viewpoints* (which relate each 2D reference drawing to the 3D base model). Given an arbitrary viewpoint, our method interpolates the key deformations to generate a 3D model that is specific to the new viewpoint, thereby capturing the view-dependent distortions of the reference drawings.

Keywords: Cartoon animation, 3D animation, rendering, animation systems, non-photorealistic rendering, 3D blending

CR Categories: I.3.5 surface and object representations; I.3.3 display algorithms

1 INTRODUCTION

Cartoon animation has continually taken advantage of developments in computer graphics. Three-dimensional elements have been used to render crowds, buildings, scenery, and even main characters. When these 3D objects are created, the modelers typically begin with a set of reference drawings of the object (the *model sheet*) showing it from different viewpoints. Unlike photographs or technical illustrations, these hand-created images do not correspond to a precise physical space – the artists who draw them try to achieve the best *aesthetic* effect, and are not bound to geometric precision. As a result, these drawings typically contain many subtle artistic distortions, such as changes in scale and perspective (also noted by [Zori95]), or more noticeable effects such as changes in the shape or location of features (e.g., the face, hair, and ears of Figure 1). Because these distortions differ in each drawing and do not correspond to a 3D geometric space, conventional 3D models are unable to capture them all. As a result, these view-specific distortions are often lost as we move from the artistic 2D world to the geometric 3D world.

One might attempt to remedy this problem using existing 3D modeling and animation tools by directly modifying the object at selected keyframes of the final animation, to match the drawings better. However, this approach is only feasible if the camera path is fixed, and might be prohibitively expensive if the object is replicated many times from different angles (e.g., in a crowd).

In this paper, we propose to make the view-dependencies an *inherent* part of the model, defining them only once during the modeling phase. The appropriate distortions can then be generated automatically for any arbitrary viewpoint or camera path.

We accomplish this with a technique we call View-Dependent Geometry – geometry that changes shape based on the direction it is seen from. A view-dependent model consists of a base model (a conventional 3D object) and a description of the model's exact shape as seen from specific viewpoints. These viewpoints are known as the key viewpoints (which are independent of the camera path that will be used during rendering), and the corresponding object shapes are the key deformations. The key deformations are simply deformed versions of the base model, with the same vertex connectivity. Given an arbitrary viewpoint or camera path, the deformations are blended to generate a new, view-specific 3D model.

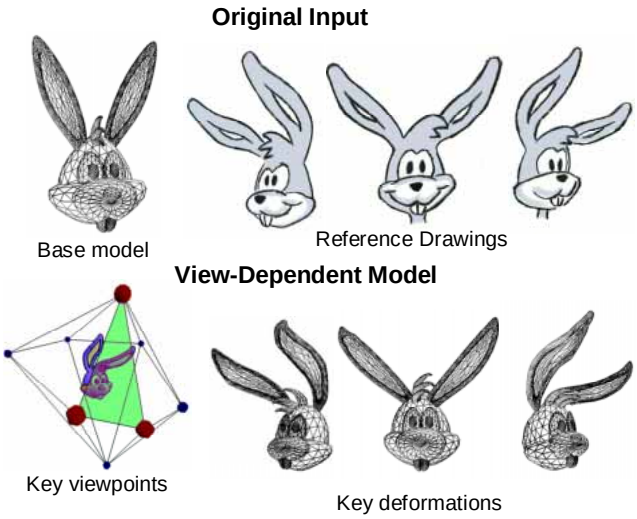

Figure 1 Example components of a view-dependent model

View-dependent models are able to capture different looks for an object from different viewing directions. By creating a key deformation for each reference drawing, we can create models that automatically respond to any given viewing direction or camera path, yielding the proper artistic distortions.

This paper describes how to construct and render view-dependent models, both static and animated. It discusses the various interpolation issues, and shows how to deal with sparse deformations. Several examples of the technique's applicability to animation are given. We note, however, that View-Dependent Geometry is not limited to the specific driving problem of artistic distortions, but rather presents a general method – that of allowing an object's shape to vary in response to view direction – which may be of use in other areas of computer graphics as well.

UNC Department of Computer Science, Sitterson Hall, CB #3175, Chapel Hill, NC, 27599-3175. Email: rademach@cs.unc.edu

2 PREVIOUS WORK

The problem of cartoon animation was addressed very early in computer graphics, for example by [Levo77, Hack77, Reev81]. Many computer systems have since been designed to aid in traditional animation. [Dura91] provides an overview of many of the issues and tradeoffs involved. [Litw91] describes the Inkwell system, based on 2-D geometric primitives. [Feke95] describes TicTacToon, which mimics the workflow of a traditional animation studio. [Robe94b] presents an overview of different commercial software packages for cartoons, and [Robe94a] describes Disney's CAPS system, the most successful usage of computers in cel animation to date. Many techniques of cartoon animation were introduced to the graphics literature by [Lass87].

The above papers describe general cel animation systems; there have also been many recent publications describing specific methods for graphics in cartoons. [Libr92] describes a system which takes as input a set of vector-based drawings, and parametrically blends them to interpolate any number of defined features. [Pfit94] describes the wildebeest scene in Disney's *The Lion King*. [Guag98] describes similar techniques in Disney's *Mulan*, as well as their enhanced 2.5-D multiplane system, Faux Plane. [Wood97] shows how to render multiperspective panoramas of backgrounds for use in cartoons. Finally, Correa demonstrates a method for integrating texture mapping with cel animation [Corr98]. Our method is similar in nature to Correa's. The common philosophy is that one should not discard the inherent expressiveness of the 2D artist as 3D computer graphics are incorporated into the animation. In their work, they apply textures to hand-drawn animated sequences (thereby retaining the artistry of the drawn animation) whereas in our method we create 3D models that are able to capture the artistic distortions in multiple drawings of an object.

Our method involves issues in 3D interpolation and deformations. Related work includes [Beie92, Lee95, Lerios95]. Our method is also similar to [Debe98], where textures – rather than geometry – are blended in response to the current viewpoint.

3 CREATING AND RENDERING VIEW-DEPENDENT GEOMETRY

A view-dependent model is composed of a base model and a set of deformations specifying the model's shape as seen from specific viewpoints. This section discusses how to determine these key viewpoints, how to construct the corresponding deformations, and how to blend between the different deformations as the object is viewed from an arbitrary location.

3.1 Creating View-Dependent Models

The inputs to our system are a conventional 3D model of an object (the base model) and a set of drawings of the object from various viewpoints (Figure 2 shows an example model created as a polygonal mesh in 3D Studio Max, and a single hand-drawn image). The base model is created using standard 3D modeling software. It represents the best-fit geometry to the complete set of drawings (since the context is artistic animation, "best-fit" can be interpreted in an intuitive – and not necessarily numerical – sense). In this section we assume the base model is rigid (animated models are discussed in Section 4).

3.1.1 Aligning the base model and the images

The first step is determining a viewpoint for each drawing. That is, we find a camera position and orientation relative to the base model, with a projection that best matches the given drawing (the current implementation uses perspective cameras, although this discussion also applies to parallel projections). Because of the artistic distortions, exact alignment of the model with the drawing is not possible. We therefore cannot use standard linear correspondence methods such as in [Faug93] (although non-linear methods might prove useful in future work). Note, however, that exact alignment is not actually necessary in our context, since the features in the 3D model that cannot be aligned will be deformed to match the image in the next step.

In the current implementation the user manually aligns the model with each drawing by rotating and translating the camera until the main features match. Figure 3 shows the model superimposed over a drawing after it has been manually aligned.

3.1.2 Deforming the base model

There are many existing techniques for deforming 3D models. For example, one may use free-form deformation lattices [Sede86], "wires" [Sing98], or various other existing methods. These all operate in 3D space, thereby permitting arbitrary changes to the object's shape.

For our application, however, full 3D deformations are not always necessary. Since we are ultimately concerned with the model's shape as seen from only a single fixed viewpoint (for each reference image), we can perform the majority of the deformations in 2D, parallel to the image plane of the camera. This makes the deformation step considerably easier than the full 3D modeling required to originally build the base model.

The current implementation of the deformation system is quite straightforward. Given a polygonal mesh representation of the model, the user picks a vertex. All vertices within a specified

Figure 2 A reference drawing of a character, and the base model. The model is constructed using standard 3D modeling software.

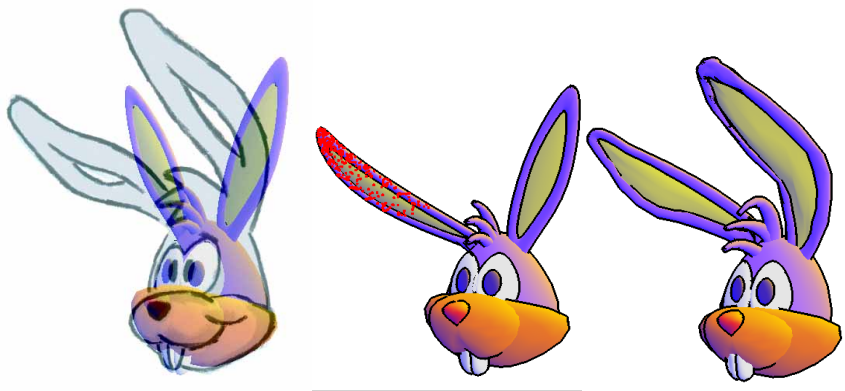

Figure 3 Construction of the view-dependent model. We first align the base model to the drawing – this establishes the *key viewpoint*. We then deform the model to match the drawing (the drawing is not altered). On the right we see the final *key deformation*.

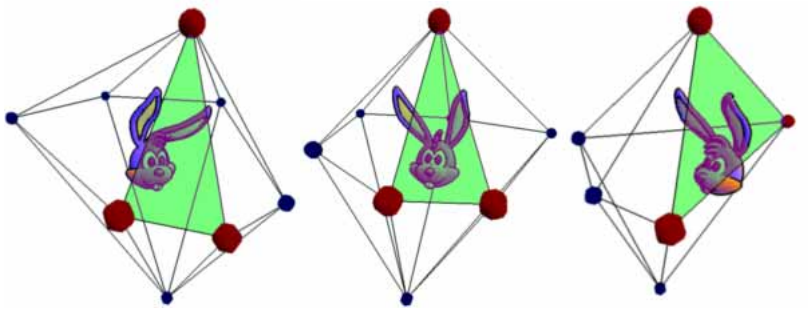

Figure 4 Viewpoints for each key deformation are shown as spheres around the model. To compute the shape as seen from the current viewpoint, we find the nearest three key viewpoints (indicated in red) and interpolate the corresponding 3D deformations.

distance ***r*** in 3D of the selected vertex are then also selected. As the user drags the original vertex across the image plane, the other vertices are dragged in the same direction with a scaling factor of $(1-d/r)^2$, where ***d*** is the distance of a given vertex to the originally-chosen point, and ***r*** is the selection radius, both in world units. The vertices' Z-distances from the camera remain unchanged. This simple method permits relatively smooth deformations (dependent on the resolution of the underlying mesh). The middle still in Figure 3 shows a deformation on the example model. The currently-selected group of vertices is shown in red.

Besides dragging vertices across the image plane, our implementation also allows the user to push and pull vertices in the Z direction (towards or away from the camera). This is necessary to fix vertices that accidentally pass through the mesh as they are translated on the image plane. A more sophisticated implementation should provide an entire suite of tools including arbitrary rotations and scalings of groups of vertices – as with any modeling application, the quality of the interface can greatly influence the quality of the resulting mesh.

Note that *the topology (vertex connectivity) of the model does not change* during the deformation; only the vertex locations are altered. This greatly simplifies the subsequent interpolation step. Also note that the *drawings are not altered* – only the base model is deformed.

The last image in Figure 3 shows the model after the deformation process. Comparing this with Figure 2, we see that the deformed model matches the reference drawing's shape more closely than the base model.

After the object and image are matched to the user's satisfaction, the deformed object is saved to file as a key deformation, and the aligned camera location is saved as a key viewpoint.

3.2 Rendering View-Dependent Models

In the previous step we saw how to specify what the model should look like when seen from specific, discrete viewpoints. These viewpoints and deformations are independent of the final camera path (they are inherent components of the model itself), and are constructed *a priori* in the modeling phase. At rendering time we need to determine – given an arbitrary camera direction relative to the model – what the object's 3D shape should be. The rendering process proceeds as follows:

1) Find the three nearest key viewpoints surrounding the current viewpoint.
2) Calculate blending weights for the associated key deformations.
3) Interpolate the key deformations to generate a new 3D model for the current viewpoint.
4) Render the resulting interpolated 3D model.

This process is similar to the view-dependent texture mapping of [Debe98], in which three textures (on a regular grid called a "view map") are blended to capture reflectance and occlusion effects in multiple images of an object.

3.2.1 Finding the nearest surrounding key viewpoints

First we must find the nearest key viewpoints surrounding the current viewpoint. In this paper we consider only the viewing *direction* for the key and current viewpoints, and not the distance from the cameras to the object (we also assume, without loss of generality, that the view directions point towards the centroid of the object). Since we do not differentiate between distances, we can map the viewing directions to points on a *viewing sphere* around the object. To find the surrounding key viewpoints, we find the three points on this sphere whose spherical triangle contains the current viewpoint (Figure 5).

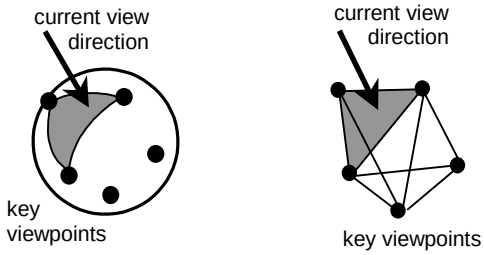

Figure 5 The key viewpoints surrounding the current viewpoint are given by the intersected spherical triangle on the viewing sphere. This is equivalent to the corresponding planar triangle on the convex hull of sphere points.

However, we can avoid working in the spherical domain by noting that a spherical triangle is simply the projection onto the sphere of the *planar* triangle between the three vertices. Therefore, if the current viewpoint maps to the spherical triangle between three given keys, it also maps to the corresponding planar triangle. This leads to the following simple method for finding the surrounding viewpoints: as a preprocess, project the key viewpoints to a sphere around the object, and then compute the convex hull of these points; this gives a triangulation around the object (any convex hull algorithm that avoids creating long, splintery triangles can be used. Our current implementation uses

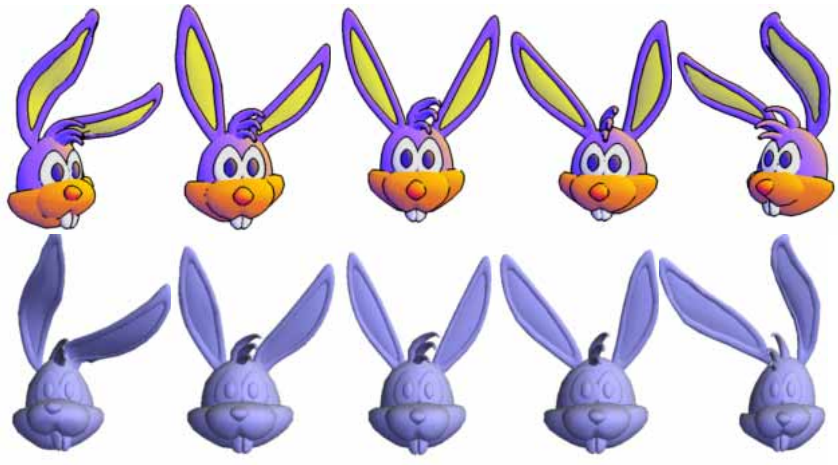

Figure 6

Top row: Different views generated as we rotate our camera about the model. They are created by interpolating the three key deformations nearest to the camera viewpoint.

Bottom row: The interpolated 3D model seen from a fixed, independent viewpoint. We clearly see the model distorting as the top row's viewpoint changes

[Wats81]). At rendering time, find the face in the convex hull which is intersected by a ray from the current camera to the sphere center. The intersected triangle denotes the three key viewpoints surrounding the current camera (note that the result of the intersection test may be zero or two points if the key viewpoints do not fully enclose the object - discussed further in Section 3.3).

Figure 4 shows the key viewpoints of our example view-dependent model, projected onto a sphere. The convex hull for these points is displayed in wireframe. For each rendered view, we project a ray from the eye towards the sphere center; the intersected triangle of the hull is shown in green. The vertices of this triangle (shown in red) correspond to the three nearest key viewpoints for the current view.

3.2.2 Calculating the blending weights

Given the intersection of a viewing ray with one of the triangles from the previous section, the blending coefficients are given directly by the barycentric coordinates w_1, w_2, w_3 of the intersection point. These weights are continuous as the camera is moved within and across triangles. Furthermore, one of the barycentric coordinates will reach a value of one – and the other two weights will equal zero – when the current viewpoint exactly matches a key viewpoint (i.e., when the intersection point is at a triangle vertex). When this occurs, the blended model will exactly match the one key deformation corresponding to that viewpoint.

The blending weights can be scaled exponentially to alter the sharpness of the transition between adjacent deformations. We define these new, scaled weights as:

$$w_i' = \frac{w_i^\alpha}{w_1^\alpha + w_2^\alpha + w_3^\alpha}$$

where α is the sharpness factor. As α grows >1, the resulting blended model moves more quickly towards the nearest key deformation. As α becomes <1, the blend is more gradual. The next section discusses the actual interpolation of the deformed models' vertices, given the three blending weights w_1', w_2', and w_3'.

3.2.3 Interpolating the key deformations

The final deformed model for the current viewpoint is generated by interpolating corresponding vertices from the three nearest key deformations. Since the key deformations all share the same vertex connectivity, the vertices correspond directly – and interpolation is a simple matter of computing a weighted blend of the corresponding vertices' positions in 3-space.

Since the interpolation is computed independently for each vertex in the final model, we can limit our discussion here to a single vertex v. We denote the vertices corresponding to v at each of the N key deformations as $\{v^1, v^2, \ldots, v^N\}$. Each of these vertices is the 3-space location of v under the deformation corresponding to one particular image. We now denote the three vertices from the nearest key deformations as v^i, v^j, v^k. Our current implementation uses a linear interpolation scheme, and thus the vertex v is given as:

$$v = v^i w_1' + v^j w_2' + v^k w_3'$$

where w_1' is the weight corresponding to the first key deformation, v^i, and similarly for w_2' and w_3'.

One could obtain smoother blends by applying higher-order interpolation. This would consist of taking the set of vertices $v^1 \ldots v^N$ and fitting a higher-order surface to it, yielding a smooth tri-patch (with the vertex connectivity given by the convex-hull triangulation). This tri-patch would be locally parameterized by two of the weights w' (since the third is always one minus the sum of the first two). Given an arbitrary viewing direction, a single point is computed on this surface and used as the current location for v.

Another interpolation scheme worth investigating for future work is that of Radial Basis Functions, a multidimensional learning-based technique. These are used by [Libr92] to generate artistic images interpolated from vector-based drawings.

Note that the above discussion only deals with the *vertex-wise* smoothness of the interpolation. We do not enforce any *global* constraints on the deformations or on the resulting interpolation (e.g., ensuring the mesh doesn't pass through itself, enforcing mesh curvature constraints, etc.). Doing so would involve a tradeoff between the quality of the resulting meshes and the extent of allowable deformations.

3.2.4 Rendering the resulting 3D model

In this paper, we display the resulting model using non-photorealistic rendering techniques, consisting of a non-standard lighting scheme and a silhouette-rendering scheme. The object is lit and shaded with a method similar in nature (though not identical) to [Gooc98]. Our lighting equation is:

$$C_{final} = k_a C_{base} + \left(\hat{l}_1 \bullet \hat{n}\right)\left(C_{warm} - \left(C_{base} * k_1\right)\right) * k_2 + \left(\hat{l}_2 \bullet \hat{n}\right)\left(C_{cool} - \left(C_{base} * k_3\right)\right) * k_4$$

where l_1 and l_2 are the positions of two lights on opposite sides of the surface, n is the surface normal, C_{final} is the final color for the

surface, C_{base} is the base color of the surface (e.g., the diffuse color), and C_{cool} and C_{warm} are cool and warm colors (e.g., blue and orange). The parameter k_a controls how much of the base color the surface receives as ambient light (the given examples use k_a = .9). The parameters k_1 and k_3 are used to prevent the lights from oversaturating the surface (since it has a strong ambient component), and k_2 and k_4 control the intensity of the warm and cool lights. A value of 1.0 can be used as a default for k_1 through k_4.

In contrast with the Gooch method, the values of k_1 through k_4 are *varied for each surface*, based on the hue of C_{base}. For example, we disable the warm (orange) light when rendering white-ish surfaces – yielding soft, cool whites. For purple surfaces, we set k_4 to 2.0 (intensifying the cool light component), while for red colors we increase the cool light while also decreasing the warm light – yielding rich purples and brilliant reds. There are many other hues that can be handled, and varying the parameters leads to many interesting effects (e.g., letting the parameters extend to negative values yields fluorescent surfaces).

The above equation can be implemented in OpenGL by setting the global ambient light to $C_{base} * k_a$, setting the per-light ambient and specular components to zero, and using the second and third terms in the above equation as the diffuse component for the first and second lights, respectively.

Our silhouette algorithm is based on [Rask99] (the techniques of [Mark97] can also been applied). In the former paper, silhouettes are generated by rendering the model in two passes: first a regular backface-culled rendering pass, with lit and shaded polygons, then a second pass with front faces culled and back faces rendered in thick black wireframe (using the OpenGL commands `glDepthFunc(GL_LEQUAL)`, `glCullFace(GL_FRONT)`, and `glPolygonMode(GL_BACK, GL_LINE)`). This method is easy to implement, and operates on unstructured "polygon soup" models as well. In our implementation, we also optimize by first detecting adjacent faces which span a silhouette edge (that is, one face is front-facing and the other back-facing). We then only render these faces – rather than all the backfaces – in the second pass. In addition, we render these faces a third time as points using `glPolygonMode( GL_BACK, GL_POINT)`, to eliminate cracks that can appear between adjacent thickened edges.

3.2.5 Example View-Dependent Model

In Figure 6 we show a static view-dependent model, rendered from a series of viewpoints. This model was created by applying the three deformations shown in Figure 1, along with 5 other minor deformations about the viewing sphere. As we rotate around the head, we see the ears and hair shift to match the original drawings (from Figure 1). In the bottom row (in blue) we show the model from an independent, fixed viewpoint. This clearly shows the 3D changes in the model as the camera rotates.

3.3 Unspecified Regions in Viewing Sphere

Our method has assumed that the key viewpoints entirely surround the base object. Then there will always be three key viewpoints surrounding any arbitrary new viewpoint. However, if the hull of the key viewpoints does not enclose the center of the viewing sphere, then there will be areas on the sphere that have no associated key deformations. This may happen, for example, if there are fewer than four deformations, or if the key viewpoints all lie on one side of the object.

We can deal with these unspecified regions by interpolating the deformations with the original base model. For example, if the key viewpoints all lie on one side of the sphere, we can insert dummy "deformations" – simply copies of the base model – in order to fully enclose it. Equivalently, we can revert to the base model without explicitly inserting dummy keys by using a cosine or cosine-squared falloff on the available deformations as the current viewpoint moves into unspecified regions.

For example, in Figure 7 we show a building model, to be used as a background object. Since backgrounds are often only seen from a single viewpoint, we only apply one deformation (from the front of the building). As the current camera moves away from the front, we gradually revert to the base model by blending the single deformation and the base model with:

$$new\_model = \max\left(0, \left(\hat{V}_{key} \bullet \hat{V}_{eye}\right)\right) * deformed\_model + \max\left(0, 1 - \left(\hat{V}_{key} \bullet \hat{V}_{eye}\right)\right) * base\_model$$

where V_{key} is the view vector for the single key deformation, and V_{eye} is the view vector for the current eye point. This formula blends smoothly between the single deformation and the base model as the viewpoint changes.

4 ANIMATED VIEW-DEPENDENT MODELS

The basic view-dependent method described above only handles static base models. This can be useful for background objects, props, vehicles, etc. In this section, we demonstrate how to deal with objects whose shape changes non-rigidly over time.

We note that if the animation will be used for only a single shot, or if the camera path for the final render is fixed, then it may be easier to match the animation and artwork directly using conventional keyframe methods. However, if we have an animation that will be used repeatedly from arbitrary angles (e.g., a walking cycle), in large numbers (e.g., crowds), or from an unknown camera path (e.g., a real-time application), then it may be more efficient to use view-dependent geometry. The animation can then be rendered from any arbitrary viewpoint, automatically yielding the proper distortions.

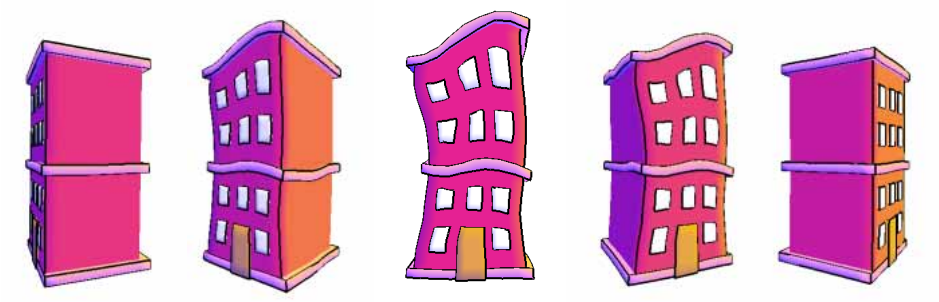

Figure 7 The base model in this example is a simple rectangular building. We applied a single deformation, from a viewpoint directly facing the building. The output model is fully deformed when the current camera faces the building front, and reverts to the original shape as the camera moves towards the sides.

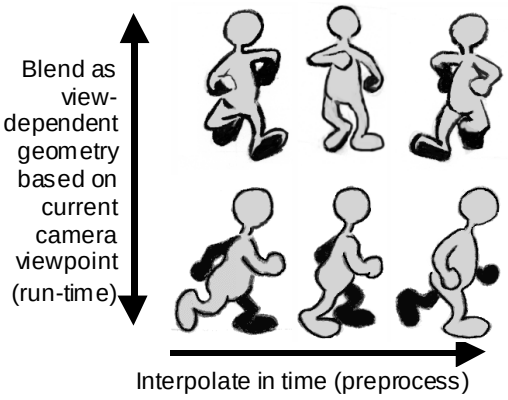

Figure 8 A set of drawings showing the object at different frames, from different viewpoints. We interpolate across time (horizontally) to generate deformations for each frame. We blend between different deformations at a given frame (vertically) to render from an arbitrary viewpoint.

4.1 Animated Base Models

A set of key deformations indicates what an object should look like from various viewpoints, but does not show change over time. When the base model is non-rigidly animated, then a single set of deformations no longer suffices. Instead, we will need a different set of deformations *for each frame of the model's animation*, which are then blended on a per-frame basis in response to the current viewpoint.

However, the user does not have to *explicitly* specify the set of key deformations for every frame – a tedious and error-prone task. Instead, the user only needs to define a small number of deformations, at different times in the animation and from various viewpoints (Figure 8). All the deformations from the same viewpoint are then interpolated over time (discussed in the next section). This yields, for each viewpoint on the viewing sphere, a deformation at every frame. Once this interpolation is applied for all key viewpoints, we will have a full set of deformations for each frame. We can then directly apply the basic view-dependent geometry method on a per-frame basis.

4.2 Interpolating the Key Deformations Over Time

Let us consider a single key viewpoint. We need a deformation of the base model from this viewpoint for every frame in the animation, but are given only a small number of them at selected frames. Because the deformations are given sparsely and do not necessarily correspond to the underlying animation keyframes, we cannot interpolate between them directly (doing so would lead to the typical problems of object blending discussed in [Beie92, Leri95, Witk95, Sede93]).

Instead of interpolating directly, we propagate the deformations throughout the underlying animation by factoring out the deformation offsets at the given frames, interpolating between these offsets, then adding the interpolated deformation offsets to the original animated model. This preserves the underlying motion while propagating the changes due to deformation.

For example, let $\boldsymbol{v}$ be a vertex in an animated base model, with deformations at two given frames. We denote $\boldsymbol{v}$'s original 3D location at the first given frame as $\boldsymbol{v_a}$, and its original position at the next keyframe as $\boldsymbol{v_b}$. We denote the *deformed* vertex locations at those two frames as $\boldsymbol{v_a'}$ and $\boldsymbol{v_b'}$. Instead of interpolating directly from $\boldsymbol{v_a'}$ to $\boldsymbol{v_b'}$, we decompose the 3D locations into:

$$\boldsymbol{v_a' = v_a + o_a}$$
$$\boldsymbol{v_b' = v_b + o_b}$$

where $\boldsymbol{o_a}$ and $\boldsymbol{o_b}$ are the offsets (3D vectors) by which the vertices are deformed in each given frame. We then interpolate the *offsets* (the current implementation uses natural cubic splines), and add each new offset vector $\boldsymbol{o_i}$ (at frame $\boldsymbol{i}$, between the two keyframes) to the undeformed vertex $\boldsymbol{v_i}$, yielding the final interpolated vertex position.

4.3 Example Animated View-Dependent Model

Figures 9-12 show an animated base model of a 40-frame walk cycle. We applied a total of 4 key deformations to the original walking model: one from the front and one from the side at frame 10, and another two from the front and side at frame 30 (these deformations simply exaggerate the pose of the model,

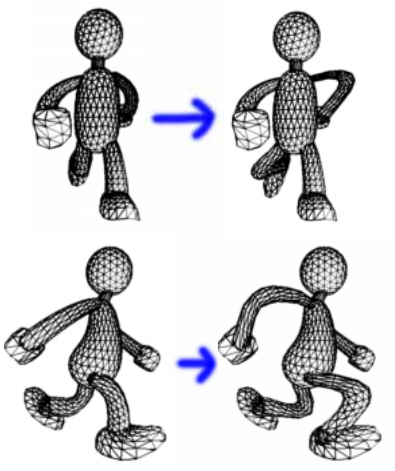

Figure 9 Two different deformations applied to an animated model. On the left is the base model, on the right is the deformed version.

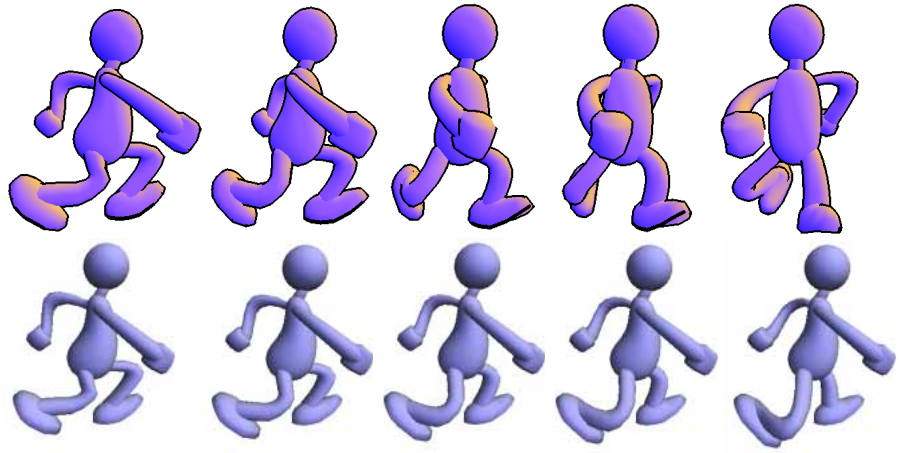

Figure 10 This sequence shows different views of the animated model, *at a single moment in the model's animation* (time is not changing, only the viewpoint). The object's shape varies depending on the viewpoint. Top row: view-dependent model as seen from the main (rotating) camera. Bottom row: model as seen from independent, fixed camera. The bottom row clearly shows the distortions in the arms and legs as the camera rotates.

making the arms and legs swing out more). We therefore have 2 key viewpoints (a front view and a side view), and two deformations in time per viewpoint (the two deformations at frame 10 are shown in Figure 9). These deformations are first offset-interpolated in time as a preprocess, yielding deformations at *every* frame, for each key viewpoint. These key deformations are then blended at run-time using the view-dependent geometry method. In Figure 10 we show a *single frame* of the walk cycle, seen from various viewpoints around the model. The blue model (seen from a fixed, independent viewpoint) shows how the object distorts as we view it from different angles. Figure 11 and 12 compare the original model against the view-dependent model as they are animated over time. Figure 11 is the original model, without view-specific distortions. Figure 12 shows the view-dependent model, clearly showing the effects of the distortions.

5 PUTTING IT ALL TOGETHER

In Figure 13 we bring together the different methods discussed in this paper. The rabbit's head is a static view-dependent model, the body is an animated view-dependent model, and the buildings are static view-dependent models with a single deformation applied. We can see the ears differ in the first and last viewpoints. We also see the distortions of Figure 9 in the middle two viewpoints. Finally, we see the buildings are distorted when seen face-on, but otherwise revert to their original rectangular shape.

6 FUTURE WORK

Both the current implementation and the general method of view-dependent geometry are open to many avenues of further investigation:

- Semi-automatic alignment and deformation based on feature correspondence. For example, one might adapt standard image morphing techniques such as [Beie92].
- Automatic construction of the 3D base model from the 2D drawings.
- Better deformations might be achieved by applying higher-order interpolation, and by ensuring vertices do not pass through the mesh as it is deformed.
- Texture extraction – It is straightforward to extract texture coordinates from each reference drawing, and then use view-dependent textures [Debe98]. However, drawings are difficult to blend due to contour lines and differences in the shading, line style or coloring of each drawing.
- Key viewpoints at different distances. Instead of a triangulation of the viewing sphere, a tetrahedralization of space would be required.
- Interface tools and model formats – The current implementation could be greatly enhanced by refining the selection tools, the deformation methods, and by operating on NURBS and other curved surfaces.

7 CONCLUSION

It is an established fact in computer graphics that the camera viewpoint plays an important role in determining the appearance of an object. From Phong shading to microfaceted reflections to view-dependent texture mapping, graphics research has shown that gaze direction is an important parameter in rendering objects. Our work extends this progression by modifying the actual *shape* of an object depending on where it is viewed from. In doing so, we directly address a problem in 3D-enhanced cel animation – the loss of view-specific distortions as an object moves from the artistic 2D world to the geometric 3D world. By employing view-dependent geometry in cartoon animation, we can render 3D models that are truer in shape to their original 2D counterparts.

ACKNOWLEDGEMENTS

The author would like to thank Gary Bishop, Susan Thayer, Nick England, Mark Mine, Michael Goslin, Gary Daines, and Matt Cutts for helpful discussions throughout the progress of this work, the reviewers for their feedback, and Todd Gaul for video editing assistance. This project was funded by DARPA ITO contract number E278, NSF MIP-9612643, DARPA ETO contract number N00019-97-C-2013, and an NSF Graduate Fellowship. Thanks also to Intel for their generous donation of equipment.

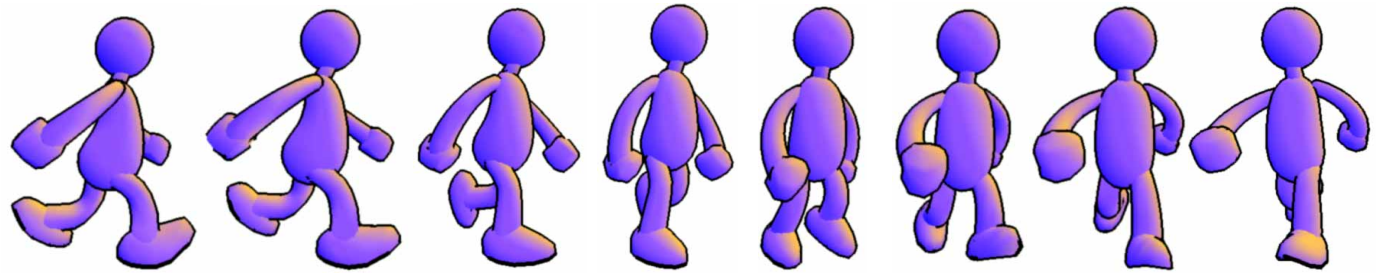

Figure 11 The base (undeformed) model animated over time, viewed by a rotating camera. At each frame, both the base model and the viewpoint changes. Since this is a conventional animated model, there are no viewpoint-specific distortions.

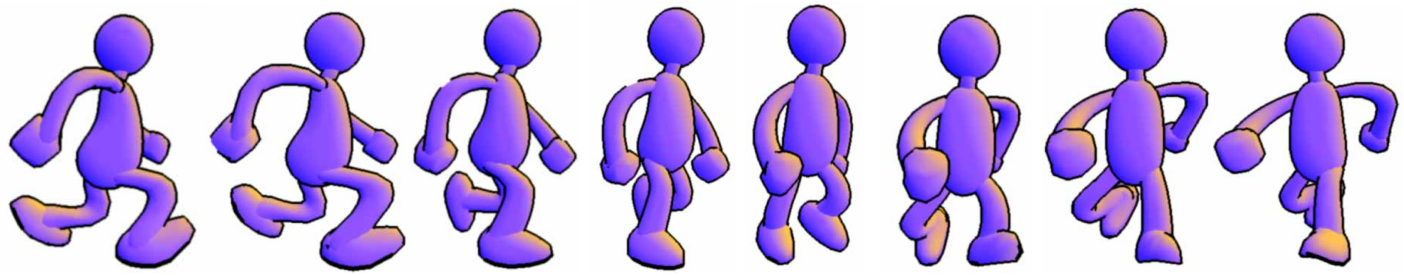

Figure 12 The view-dependent model animated over time, viewed by a rotating camera. The animated model's shape changes based on the viewpoint. The distortions of Figure 9 are seen in the 2nd and 7th frames – all other frames use offset-interpolated deformations.

REFERENCES

[Beie92] Thaddeus Beier and Shawn Neely. Feature-Based Image Metamorphosis. In *Proceedings of SIGGRAPH 92*, pages 35-42. New York, July 1992. ACM.

[Blai94] Preston Blair. *Cartoon Animation*. Walter Foster Publishing, Laguna Hills, California, 1994.

[Corr98] Wagner Toledo Correa, Robert Jensen, Craig Thayer, and Adam Finkelstein. Texture Mapping for Cel Animation. In *Proceedings of SIGGRAPH 98*, pages 435-446. New York, July 1998. ACM.

[Debe98] Paul Debevec, George Borshukov, and Yizhou Yu. Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. In *9th Eurographics Rendering Workshop*, Vienna, Austria, June 1998

[Dura91] Charles Durand. The "Toon" Project: Requirements for a Computerized 2D Animation System. In *Computers and Graphics* 15 (2), pages 285-293. 1991.

[Faug93] Olivier Faugeras. *Three-Dimensional Computer Vision: A Geometric Approach*. MIT Press, Cambridge, Massachusetts, 1993.

[Feke95] Jean-Daniel Fekete, Erick Bizouarn, Eric Cournarie, Thierry Galas, and Frederic Taillefer. TicTacToon: A Paperless System for Professional 2D Animation. In *Proceedings of SIGGRAPH 1995*, pages 79-90. July 1995. ACM.

[Gooc98] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A Non-Photorealistic Lighting Model for Automatic Technical Illustration. In *Proceedings of SIGGRAPH 98*, pages 447-452. New York, July 1998, ACM.

[Guag98] Eric Guaglione, Marty Altman, Kathy Barshatzky, Rob Bekuhrs, Barry Cook, Mary Ann Pigora, Tony Plett, and Ric Sluiter. The Art of Disney's Mulan. In *SIGGRAPH 98 Course Notes #39*. New York, July 1998. ACM.

[Hack77] Ronald Hackathorn. Anima II: a 3-D Color Animation System. In *Proceedings of SIGGRAPH 77*, pages 54-64. New York, 1977. ACM.

[Lass87] John Lasseter. Principles of Traditional Animation Applied to 3D Computer Animation. In *Proceedings of SIGGRAPH 87*, pages 35-44. New York, July 1987. ACM.

[Lee95] Seung-Yong Lee, Kyung-Yong Chwa, Sung Yong Shin, and George Wolberg. Image Metamorphosis Using Snakes and Free-Form Deformations. In *Proceedings of SIGGRAPH 95*, pages 439-448. New York, July 1995. ACM.

[Leri95] Apostolos Lerios, Chase Garfinkle, and Marc Levoy. Feature-Based Volume Metamorphosis. In *Proceedings of SIGGRAPH 95*, pages 449-456. July 1995. ACM.

[Levo77] Marc Levoy. A Color Animation System Based on the Multiplane. In *Computer Graphics*, vol. 11, pages 65-71. New York, July 1977.

[Libr92] Stephen Librande. Example-Based Character Drawing. M.S. Thesis, MIT. September 1992.

[Litw91] Peter Litwinowicz. Inkwell: A 2½-D Animation System. In *Proceedings of SIGGRAPH 91*, pages 113-122. New York, July 1991. ACM.

[Mark97] Lee Markosian, Michael Kowalski, Samuel Trychin, Lubomir Bourdev, Daniel Goldstein, and John Hughes. Real-Time Nonphotorealistic Rendering. In *Proceedings of SIGGRAPH 97*, pages 415-420. July 1997. ACM.

[Pfit94] Gary Pfitzer. Wildebeests on the Run. In *Computer Graphics World*, pages 52-54. July 1994.

[Rask99] Ramesh Raskar and Michael Cohen. Image Precision Silhouette Edges. To appear in *Proceedings of Interactive 3D Graphics 99*.

[Reev81] W. T. Reeves. Inbetweening For Computer Animation Utilizing Moving Point Constraints. In *Proceedings of SIGGRAPH 81*, pages 263-269. July 1981. ACM.

[Robe94a] Barbara Robertson. Disney Lets the CAPS Out of the Bag. In *Computer Graphics World*, pages 58-64. July 1994.

[Robe94b] Barbara Robertson. Digital Toons. In *Computer Graphics World*, pages 40-46. June 1994.

[Sede86] Thomas Sederberg and Scott Parry. Free-Form Deformation of Solid Geometric Models. In *Proceedings of SIGGRAPH 86*, pages 151-160. New York, Aug 1986. ACM.

[Sede93] Thomas Sederberg, Peisheng Gao, Guojin Wang, and Hong Mu. 2-D Shape Blending: An Intrinsic Solution to the Vertex Path Problem. In *Proceedings of SIGGRAPH 93*, pages 15-18. New York, July 1993. ACM.

[Sing98] Karan Singh and Eugene Fiume. Wires: A Geometric Deformation Technique. In *Proceedings of SIGGRAPH 98*, pages 405-414. New York, July 1998. ACM.

[Wats81] David Watson. Computing the N-Dimensional Delaunay Tessellation With Application to Voronoi Polytopes. In *The Computer J.*, 24(2), p. 167-172. 1981.

[Witk95] Andrew Witkin and Zoran Popovic. Motion Warping. In *Proceedings of SIGGRAPH 95*, pages 105-108. New York, July 1995. ACM.

[Wood97] Daniel Wood, Adam Finkelstein, John Hughes, Craig Thayer, and David Salesin. Multiperspective Panoramas for Cel Animation. In *Proceedings of SIGGRAPH 97*, pages 243-250. New York, July 1997. ACM.

[Zori95] Denis Zorin and Alan Barr. Correction of Geometric Perceptual Distortions in Pictures. In *Proceedings of SIGGRAPH 95*, pages 257-264. July 1995. ACM.

Figure 13 We combine the various techniques into one scene. The rabbit's head is a static view-dependent model. Note the different tilt of the ears between the first and last viewpoints. The body is an *animated* view-dependent model. It incorporates the view-specific distortions of Figure 9. The background buildings are view-dependent models with a single key deformation. They distort when seen front-on, and appear rectangular as the viewing angle increases.