Non-linear Perspective Widgets for Creating Multiple-View Images

Nisha Sudarsanam* Mindjet Corporation

Cindy Grimm† Washington University in St. Louis

Karan Singh‡ University of Toronto

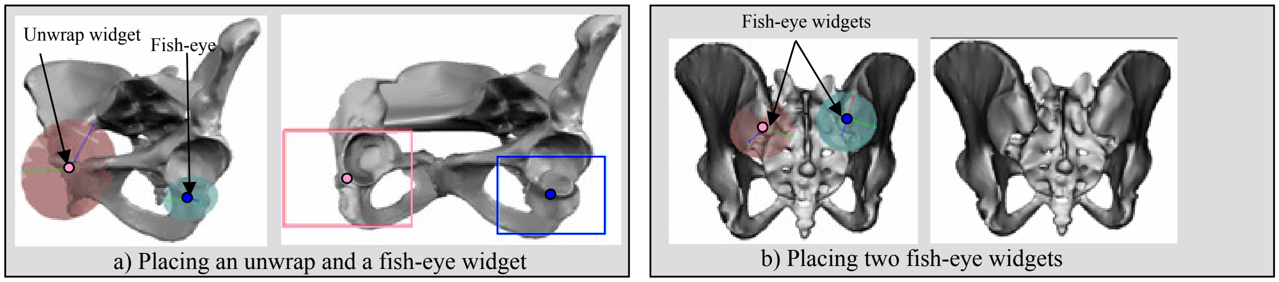
Unwrap widget
Fish-eye
a) Placing an unwrap and a fish-eye widget
Fish-eye widgets
b) Placing two fish-eye widgets

Figure 1: *Using multiple non-linear widgets to compare features on a human pelvis. (a) The unwrap widget is first used to place the two acetabulum cavities in the same views. Next, we add a fish-eye widget to the left cavity to aid the comparison (1,289,814 faces, 49,989 vertices). (b) Placing two fish-eye widgets, one for each joint in the pelvis.*

Abstract

Viewing data sampled on complicated geometry, such as a helix or a torus, is hard because a single camera view can only encompass a part of the object. Either multiple views or non-linear projection can be used to expose more of the object in a single view, however, specifying such views is challenging because of the large number of parameters involved. We show that a small set of versatile widgets can be used to quickly and simply specify a wide variety of such views. These widgets are built on top of a general framework that in turn encapsulates a variety of complicated camera placement issues into a more natural set of parameters, making the specification of new widgets, or combining multiple widgets, simpler. This framework is entirely view-based and leaves intact the underlying geometry of the dataset, making it applicable to a wide range of data types.

CR Categories: I.3.3 [Computing Methodologies]: Computer Graphics—Picture/Image GenerationViewing algorithms;

1 Introduction

One advantage of computers is that we can interact with, and display, complex models. To date, linear projection is the most common method for producing 2D images of 3D models. However, linear projection has its limitations [Glassner 2004] — while it is a good approximation of the human visual system, it can be difficult to find the "best" view of a model, or even to see certain features of a model in a single view. Traditional artists such as M.C. Escher, David Hockney and Picasso realized this, and deliberately introduced distortions of perspective in their work in order to create artistic effects, mood changes, and to control the composition of a scene. The perspective distortion introduced in these examples still relied on traditional linear perspective, but only locally. Singh [Singh 2002] referred to these distorted perspectives as "non-linear perspectives", and showed that computers can produce similar distortions.

From a conceptual stand-point, non-linear or multiple perspective images are built by blending two (or more) local linear perspectives together. Often an image has some notion of a "global" perspective, with the local perspectives defined as changes to this global one. This global perspective helps to provide the user with some idea of the 3D arrangement of the elements of the scene.

From an implementation stand-point there are two aspects to producing non-linear perspectives. The first is the camera model or mathematics used to produce them, the second is how the user interacts with the camera model [Brosz et al. 2007]. Most of the existing proposed camera models are very general and very powerful — but also difficult or time-consuming for the user to use.

In this paper we focus on making interaction very simple and easy, trading versatility for usability. We take a two-tiered approach. At the bottom is a general-purpose framework which encapsulates four key ideas: What part of the model should be displayed, how should its perspective be altered, where should it be displayed on the image, and how should this local perspective interact with other existing ones. Above this we have defined four specific widgets which were motivated, in part, by specific visualization tasks (Figures 1 and 2).

Like many approaches, we specify the projection by defining a global viewpoint with local distortions. Unlike existing approaches, our local distortions are defined so that they change appropriately as the global viewpoint changes (Figures 3 and 9). This is the key to making interactive non-linear projections — the user can still interact with, and change, the global view while keeping the desired local distortions.

Paper organization: We start with related work. Next, we describe the choices behind the underlying framework and the four widgets defined on top of it. We follow this with implementation specifics

*e-mail: nisha.sudarsanam@gmail.com
†e-mail: cmg@wustl.edu
‡e-mail: karan@dgp.toronto.edu

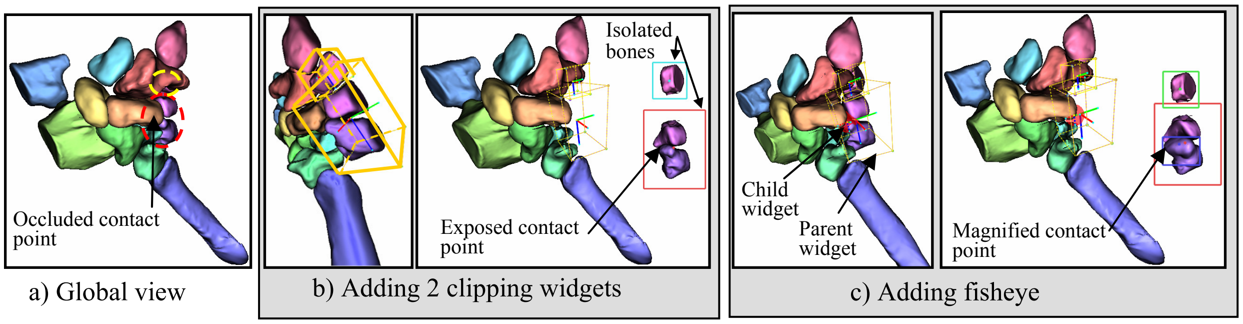

Figure 2: *(a) Our goal is to expose both the bone lying within the yellow oval and the contact points between the bones in the red oval. Note that both of these features are currently completely occluded. (b) We add two clipping widgets in order to expose all three bones. (c) To magnify the contact point between the two bones we add a fish-eye widget as a child widget of the clipping widget.*

(the framework, the individual widgets, and how to combine multiple widgets). Finally, we close with results.

1.1 Contributions

We present several widgets, each of which encapsulates a specific composite projection into a simple-to-use, but still flexible, interface. These widgets can be combined to create more complicated projections. Unlike most non-linear projection systems, the user is free to change the global view — the resulting composite view is updated in a meaningful way.

The widgets presented here are built on top of a real-time, general, non-linear perspective rendering framework. We demonstrate that this framework easily supports existing projections by creating a fish-eye and multi-perspective panorama widget.

2 Related Work

Non-linear perspectives and geometrical deformations are very closely related in their effects, but differ in their implementation. In general, view-based approaches can be applied to almost any data set and do not alter that data set. In contrast, geometric deformations are often tied to the underlying model representation and actually alter the model.

2.1 Deformations

Rademacher et al. [Rademacher 1999] and Martin et al. [Marttin et al. 2000] present approaches for deforming geometry based on the observer's position. Rademacher et al. [Rademacher 1999] blend a set of pre-defined object-deformations corresponding to a set of viewpoints closest to the current viewpoint. Martin et al. [Marttin et al. 2000] use observer-dependent control functions to indirectly control the transformations applied to the model. Our approach is most like the latter, although our set of deformations is very different.

Various approaches exists for interactive viewing of volume data sets or geographical maps [Takahashi et al. 2002]. These methods deform the underlying geometry of the data set in order to expose interesting structures ([Bruckner and Gröller 2005; Grimm et al. 2004; McGuffin et al. 2003]). These deformations depend on the current view of the model so changing the view of the model results in an inconsistent deformation, or the user must define a new deformation.

Yagel et al. [Kurzion and Yagel 1997] present an alternative approach to deforming a model that consists of deformation proxies or rendering agents which were inspired by traditional object-deformation paradigms. These rendering agents are view-independent. That is, irrespective of the view, the object appears to be deformed in the same way as it was initially defined. Our deformations are similar, except that we also guarantee that the selected region will always remain visible to the viewer.

Carpendale [Carpendale et al. 1997] developed a 3D technique for "pushing" occluding objects out of the line of sight to reveal one (or more) focus objects. Like our approach, this deformation is dependent upon the viewing direction, so the user is free to change the global view while keeping the focus objects revealed.

2.2 Non-linear perspectives

The field of non-linear perspective cameras is growing; a recent paper [Brosz et al. 2007] provides a good overview of existing models and shows how many of these models can be described using a view-based, free-form deformation. We focus here on the subset of approaches most related to our work.

Fish-eye or other types [Yang et al. 2005] of lenses are extremely useful for visualization of information graphs and other applications. Magnification lenses [Wang et al. 2005; LaMar et al. 2001] are one method of visualizing expanded views of volume-data. We include a widget for performing magnifications of specific areas of the model.

An attractive method for defining cameras is to use image-space constraints [Blinn 1988; Gleicher and Witkin 1992; Coleman et al. 2005]. Unfortunately, it can be difficult to control the camera in this manner because the solver is not guaranteed to produce a meaningful solution. Instead, we turn to defining specific camera changes which produce known image-space changes [Sudarsanam et al. 2005; Grimm and Singh 2005].

Initial work on multi-projection techniques focused on rendering individual objects from different view points and compositing the results together [Agrawala et al. 2000; Grimm 2001]. This was extended to multiple cameras altering a single, global view [Singh 2002] with techniques to maintain global scene coherence [Coleman and Singh 2004]. These are very general systems and require a lot of user input. We focus instead on creating a limited set of specific multi-projections that are very quick and easy to define.

Multi-perspective panoramas can be produced from either a 3D model and a camera path [Wood et al. 1997] or a set of images [Szeliski 1996; Agarwala et al. 2006]. A panorama is really a special kind of non-linear projection. We show how our toolkit can

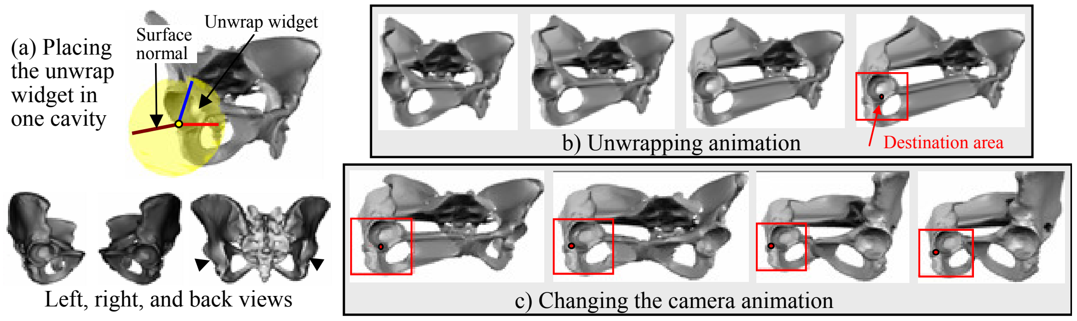
(a) Placing the unwrap widget in one cavity
Surface normal
Unwrap widget
b) Unwrapping animation
Destination area
Left, right, and back views
c) Changing the camera animation

Figure 3: *Showing the left and right acetabulum cavities in one image using the unwrap widget. (a) User places the widget in one cavity. (b) Cavity is rotated and placed in the destination area (which is automatically calculated). (c) User changes the global camera to bring the other cavity into view.*

be used to create panoramas from known 3D geometry and a set of keyframes.

3 Approach

3.1 The framework

Non-linear perspectives have traditionally been defined by specifying several local perspectives and blending them together [Coleman and Singh 2004; Coleman et al. 2005; Yang et al. 2005] or by deforming an existing view [Brosz et al. 2007; Popescu et al. 2006; Sudarsanam et al. 2005]. Our framework combines elements of both approaches, but is model and image-based instead of entirely view-based — what part of the model do you want to display, where do you want to display it, how do you want to change the viewing parameters, and how do you want to blend this view with existing ones? The key thing to note is that our local views are defined as *changes* to the current global view, so that if the user changes the global view, the local views change in a meaningful manner.

To define a widget in this framework the widget must define the following four elements:

- A 3D source volume. This is the region of 3D space the new local camera will affect.
- A 2D destination area. This is the region of image space the 3D source volume will be projected into.
- A set of camera parameter changes that distort the current global view into the desired local perspective.
- A 3D blend region, which is a (possibly empty) subset of the 3D source volume. The local perspective will be blended into the global one in this area.

The framework is responsible for blending the local perspectives together based on the blend regions. The motivation behind this design is two-fold: First, individual widgets can be defined, and used, independently of each other, and it is relatively simple to build composite widgets that communicate through the source and destination areas (Section 4.3). Second, the user can manipulate both the widget *and* the global view, and the local perspective will adapt accordingly.

Finally, our framework makes no assumption regarding the representation of the data. The only constraint we require is that the data be sufficiently sampled. If, for example, meshes are used to represent the data, then the triangulation should be sufficiently dense. For volume data, the voxel size needs to be sufficiently small relative to the amount of deformation.

3.2 The widgets

We have developed four widgets that each encapsulate a specific aspect of a non-linear perspective change. The *unwrap* widget allows the user to select a portion of the model and view it from a specific direction, regardless of the orientation of the global camera. The *clipping* widget allows the user to select a portion of the model that is occluded and pull it out of the model so it is no longer occluded. In this case, the view direction tracks the global one so that the user can see both the inside and the outside of the object from the same direction. The *fish-eye* and *panorama* widgets are versions of the familiar fish-eye zoom and multi-perspective panorama.

4 Implementation

The framework takes in a 3D model and maps it to the image plane by mapping each vertex [1] independently. A vertex's image plane location is a blended combination of the widgets that operate in that area and, possibly, the global camera. More specifically, each widget defines an area of influence and a local camera.

4.1 The framework

We first discuss what each widget needs to define, then how the framework uses this information to project a vertex.

Each widget must define the source volume and the region over which its local camera should be blended with the global one. To allow the framework to perform the blending, each widget defines a function $w : \mathbb{R}^3 \rightarrow [0,1]$ that takes in a point Q and returns a number between zero (not in the source volume) and one (in the source volume).

For the destination area, the widget defines a 3D bounding box around its region of interest and a desired center and size in the image plane. The framework automatically adjusts the center of projection and zoom of the widget's local camera so that the 3D bounding box is projected into the destination area.

Finally, the widget must define a local camera $C(Q) : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ to use for projecting the vertices. This camera is defined in terms of changes to the global camera $G : \mathbb{R}^3 \rightarrow \mathbb{R}^2$.

Using these three elements, the framework projects a vertex using the following equation:

$$P(Q) = (1 - \sum_{k=1}^{n} w_k(Q))G(Q) + \sum_{k=1}^{n} w_k(Q)C_k(Q) \quad (1)$$

[1] If we are doing volume rendering this is the grid vertices of the volume.

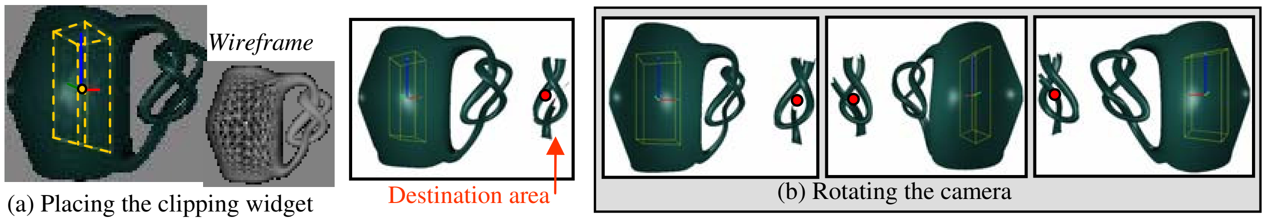

Figure 4: *This mug (5382 vertices, 10768 faces) has a knot on the inside as well as a knot in the handle. (a) Placing the clip widget around the interior knot. We include a wireframe image to show the inside knot. (b) As the global camera rotates the local view of the internal knot rotates in the same way, allowing comparison between the two knots. The destination position is recalculated for each frame.*

If the sum of the weight functions w_k is greater than one then we normalize by dividing each weight by the sum of the weights. We do not normalize if the sum is less than one because we want to blend with the global camera. Note that if there is only one local camera then this equation reduces to blending between the local camera and the global one based on w, as expected.

As an aside, we could blend the individual camera parameters and then project the vertex using this blended camera. While this method might be considered the "correct" one, it is time-consuming. We found that Equation 1 is a good approximation which works well in practice. Essentially, we are blending between locations in the 3D projection volume instead of blending the parameters that define the projection volume.

4.1.1 Source volumes and blend region

To simplify the specification of the source volumes and blend regions we define a default fall-off function that takes as input three parameters, r_{in}, r_{out}, and O, representing the inside and outside radii and center of the volume. The fall-off function returns one if the point is inside the inner sphere, and fades to zero outside of the outer sphere.

$$g(x) = (x^2-1)^2 \tag{2}$$

$$w(Q) = \begin{cases} 1 & \|Q-O\| < r_{in} \\ g\left(\frac{\|Q-O\|-r_{in}}{r_{out}-r_{in}}\right) & r_{in} \le \|Q-O\| \le r_{out} \\ 0 & \|Q-O\| > r_{out} \end{cases} \tag{3}$$

By default, r_{in} is set to $0.8r_{out}$. This strikes a balance between creating a smooth blend and minimizing the amount of the model that is rendered distorted [Zanella et al. 2002]. Decreasing r_{in} increases the size and visual smoothness of the transition region, but also decreases the amount of the model that is mapped with just the local camera.

4.1.2 Destination area

The framework can automatically calculate a zoom and center-of-projection change to move and re-size the destination area. Each widget supplies a 3D bounding box B that serves as a proxy for the source volume, and a current camera projection C. Before adjustment the destination area is the 2D bounding box around $C(B)$, i.e., the 3D bounding box projected to the screen using C. The current 2D bounding box can be moved and re-sized to the desired one using a center-of-projection (COP) and focal length change.

Let (v_x, v_y) be the vector from the current bounding box to the desired one. To shift the box, we add (v_x, v_y) to C's current COP. The focal length adjustment is determined using similar triangles and the 3D bounding box [Grimm and Singh 2005].

4.2 The widgets

4.2.1 The unwrap widget

The unwrap widget addresses the problem of trying to see two sides of a model at the same time (see Figures 3 and 9). Although this can always be accomplished using two images, it can be hard to mentally integrate where the two features are spatially. Using the unwrap widget, the user can fix one feature's view point then interactively [2] move the global view point to bring the other one into view. This interaction helps to establish the spatial relationship between the two features while still viewing them from "good" angles.

To specify the source volume the user places a spherical 3D widget around the area of interest. This widget has handles for controlling the 3D position and scale (r_{out}) of the selected volume. The surface normal closest to the center of the widget defines the default orientation of the widget (Figure 9); this can be overridden if desired.

The fall-off function is calculated as defined in the previous section. The local camera is constructed from the global one by rotating the view point to look down outward-pointing direction of the unwrap widget.

Our goal is to automatically place the selected region in screen space so that the it does not overlap the model — note that the user can override this default if desired. We first calculate what part of the screen p_m is currently occupied by the model. This is found by projecting the model's 3D bounding box using the global camera. We next determine if the projected source volume p_v lies to the left or right of the center of p_m. We push p_v out of p_m in this direction, then scale p_v (if necessary) so that it fits on the screen.

It is possible that there is not enough space around p_m for p_v. In this case, the user can either choose to have the destination area overlap p_m, or have the system find a pan and scale for the *global* camera that ensures that p_m does not occupy more than $2/3$ of the horizontal screen space.

4.2.2 The clipping widget

Sometimes there are internal structures in a model which are not visible from any direction. Like a traditional cut-away, the clipping widget supports viewing these structures without the intervening geometry; unlike the traditional approach, the viewing angle follows the global view, making it easy to see both the inside and the outside from a variety of views.

The user specifies the source volume using a bounding box, which they scale and place inside the model (Figure 4(a)). In this widget

[2] This interactivity is difficult to convey with static images; please see the video.

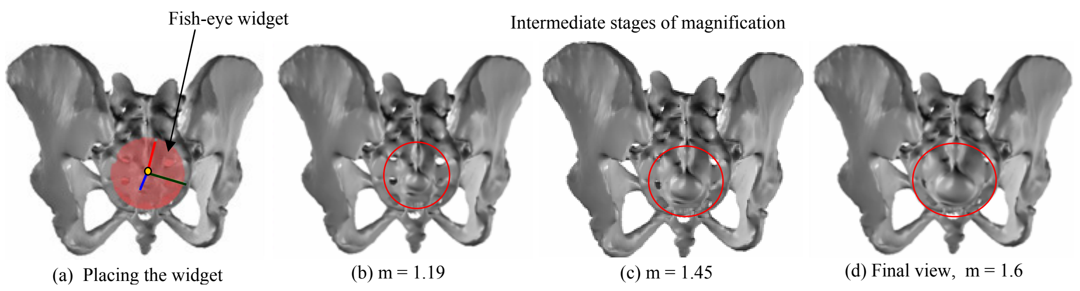

Figure 5: *(a) The fish-eye widget is represented as a sphere. Note the increasing size of the joint in successive frames (b-d) of the magnification.*

the fall-off region is empty — the vertices are either projected with the global camera or the default one (inside the bounding box). The default camera is identical to the global one, except the near and far clipping planes are adjusted to enclose the bounding box.

The destination area is calculated as for the unwrap widget, described above.

4.2.3 The fish-eye widget

The fish-eye widget (Figure 5) is represented by two concentric spheres, which represent the inner (r_{in}) and outer (r_{out}) boundaries of the source blend volume. Inside of the inner sphere the model is magnified by a factor m, provided by the user. The destination area is the same as the original, i.e., there should be no pan and zoom.

The camera change can be accomplished either with a focal length f change (makes the region bigger) or by panning the camera in, which also introduces a perspective change. In the latter case, the COP must be adjusted so that the destination area stays the same (O is the center of the fish-eye widget, $C^{'}$ is the camera projection after panning):

$$Eye^{'} = Eye + \vec{Look}(f(1.0 - 1.0/m)) \quad (4)$$
$$COP^{'} = COP + (C(O) - C^{'}(O)) \quad (5)$$

4.2.4 Panorama

The user defines a panorama by providing a set of keyframes, similar in spirit to Wood et. al. [Wood et al. 1997]. Because our panorama approach is based on blending geometry, and not images, we can support keyframes with substantial camera changes (see Figure 7). Our panorama is also very similar to the view-based approach of Singh [Singh 2002].

Unlike the previous widgets, the panorama widget produces multiple source volumes, local cameras, and destination areas, one for each keyframe. The source volumes are created by partitioning the model up amongst the different keyframes, and blending between the partitions.

The source volumes are created by assigning each vertex in the model to the camera that it is "closest" to. For each camera k, cast a ray from the eye through the middle of the film plane and find the first intersection point P_k with the model. For each vertex, find the point P_k that it is closest to and assign it to that camera. The clusters for the face panorama can be seen in Figure 7(b).

Note that in image-based rendering approaches it is more common to use the dot product between the surface normal and the view

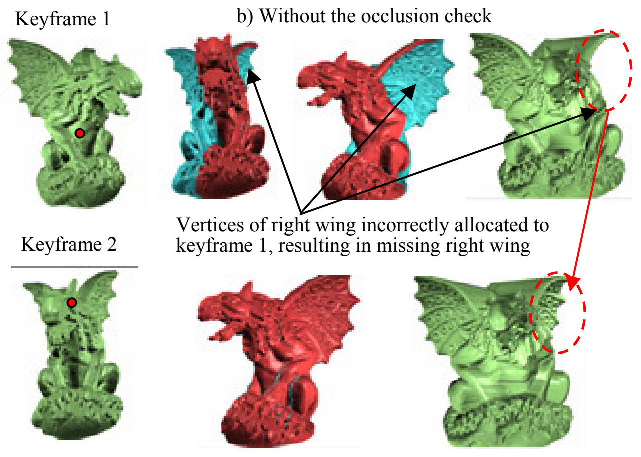
Keyframe 1
b) Without the occlusion check
Vertices of right wing incorrectly allocated to keyframe 1, resulting in missing right wing
Keyframe 2
a) Input keyframes
c) With the occlusion check

Figure 6: *(a) The two input keyframes. The right wing is occluded in keyframe 1 but not in keyframe 2. (b) The right wing is closer to the center of camera 1's ROI than camera 2's, so it is initially assigned to camera 1, which results in a missing right wing. (c) After checking for occlusion the right wing is correctly assigned to camera 2 and appears in the final panorama.*

direction to determine camera assignment. However, in general the surface normal of our models tends to vary a lot, which results in a large number of small, isolated regions. This is why we do not use this approach.

While our approach does tend to produce large, connected regions, it is possible for vertices that are visible in one view to incorrectly be assigned to a different view where they are occluded (Figure 6). This occurs since our metric for clustering vertices does not take into account the visibility of a vertex. To fix this problem, we can incorporate an occlusion test. Create an id buffer for each camera, and only assign a vertex to that camera if it is visible.

The weight function $w_k(Q)$ for each keyframe is constructed by finding a pair of cameras that the point Q lies between and defining r_{out} as the distance between those cameras. If there is no such pair of cameras then $w_k(Q)$ is set to one for the camera k that Q was assigned to; all other weights are set to zero.

To determine "between", let E_k and P_k be the eye point and center point of the camera assigned to Q. Let P_i be the center point of another camera. If $< Q - P_k, P_i - P_k >$ is positive, then Q lies between P_i and P_k. We take the closest such keyframe and define

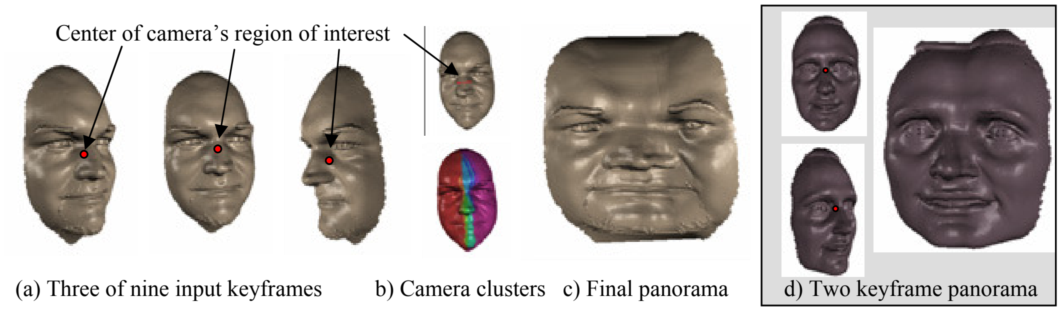
Center of camera's region of interest
(a) Three of nine input keyframes
b) Camera clusters
c) Final panorama
d) Two keyframe panorama

Figure 7: *Using nine (a) and two (d) keyframes to create a panorama that "unfolds" the face (7256 vertices, 14271 faces) across the image plane. (a) Illustrates three of the nine keyframes used to generate (c). (b) Shows the mesh colored by which camera the vertices belong to. (c) The final panorama. (d) Using only two key frames.*

w_k and w_i as follows:

$$r_{out} = \|P_i - P_k\|/2 \quad (6)$$
$$r_{in} = r_{out}/2 \quad (7)$$
$$d = < Q - P_k, P_i - P_k > /(2r_{out}) \quad (8)$$
$$w_k(Q) = \begin{cases} 1 & d < r_{in} \\ g(\frac{d-r_{in}}{r_{out}-r_{in}}) & r_{in} \leq d \leq r_{out} \\ 0 & > r_{out} \end{cases} \quad (9)$$
$$w_iQ = 1 - w_k(Q) \quad (10)$$
$$(11)$$

and all other camera weights for Q are set to zero.

To find the destination areas, first find the average C_a of all of the keyframes by averaging their parameters. Project all of the P_k by C_a to define the destination centers for each camera; the sizes are left unchanged. Moving these positions changes the span of the panorama (Figure 11).

4.3 Multiple widgets

The framework makes it simple to combine single widgets into more complex ones (see Figures 1, 2 and 10). We define three types of relationship that can exist between multiple widgets placed in a data set: Independent, parent-child, and chained. The relationship determines how the widgets communicate, particularly with regards to the destination area. While placing the widgets, the user can specify one of these relationships, with "independent" being the default choice.

Independent widgets: This is the simplest case, where multiple widgets are placed in a scene independently of each other. Each widget has its own region of influence, which may or may not overlap with another. We compute the destination area independently for each widget as per the approach outlined in the previous sections. If the destination areas map to the same side of the object the system will arrange the 2D bounding boxes vertically as in Figure 1.

Parent-child widgets: In this case, widgets are usually embedded spatially within one another, so that the child's source volume is contained within the parent's. For example, a child fish-eye widget could be embedded within a parent unwrap widget for magnifying part of an unwrapped region. When the user selects such a relationship, the framework passes the *parent's* local camera to the child widget instead of the global one. Thus, the fish-eye widget's camera is created using the rotated camera of the unwrap widget. The child's destination area is also set to be the parent's.

Chained widgets In this case, multiple widgets (usually of the same type) are strung together. Chaining widgets is useful in order to specify a complicated piece of geometry or to prevent a fold-over of geometry. When multiple widgets are chained together then the calculation of the destination area follows the standard approach only for the first widget in the sequence. The center-of-projection (or zoom) is then propagated down the chain. Thus, the after-projection parameters of successive widgets are computed with respect to earlier widgets in the chain. This is done so that widgets appear sequentially on the screen.

Foldover of geometry: Foldover or self-intersection of geometry occurs when the difference between the default camera and the local camera is large. In such a case, different faces in the transition region overlap one another resulting in self-intersections in the transition region. No amount of blending can prevent this if the two views are really disparate. The solution is for the user to place additional smaller widgets that individually result in fold-over free transformations. These smaller widgets exhibit a chain dependence on each other.

4.4 Adding additional widget types

To add an additional widget to the framework the implementor needs to create the widget itself and a function which takes in the standard camera parameters (position, orientation, focal length, center-of-projection) and produces the new, local camera parameters. The system will, by default, use the 3D bounding box containing the widget to specify the source volume (r_{in}) and the initial projected destination area. All of these defaults can be over-ridden.

4.5 GPU Implementation details

The calculation of the source volume, destination areas, and local cameras are performed on the GPU. The actual movement of the vertices is implemented on the GPU as a vertex program. The vertex program takes in the weights, local camera matrices, and the global camera matrix. It outputs the blended output location (Equation 1) and the vertex position and normal projected with just the global camera matrix. The latter is used to perform lighting.

The GPU (GeForce Go 6400) implementation is substantially faster than a pure CPU (1.73Ghz Pentium M processor) one, running at 0.78 frames per second rather than over 130 seconds for the pelvis model with one widget. The rendering time is dependent on the number of widgets and vertices but is independent of the type of widgets introduced in the scene.

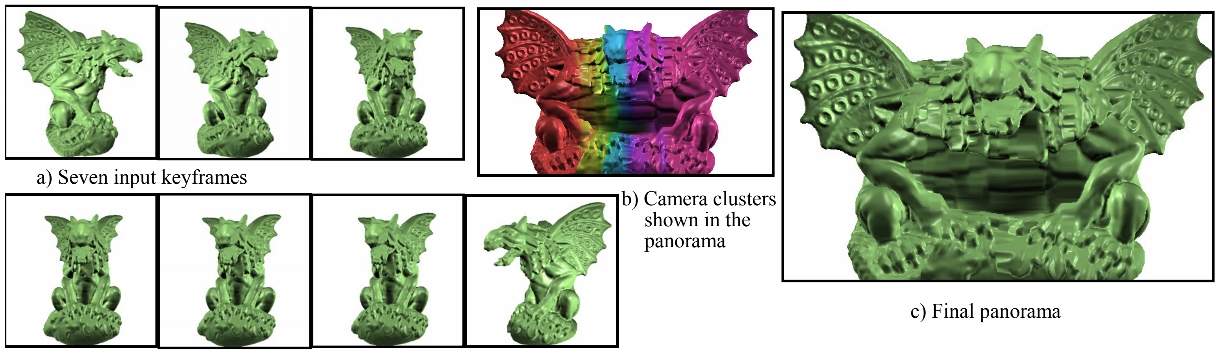

Figure 8: *(a) Seven of the eleven key frames used to generate a panorama of the gargoyle (129722 vertices, 259440 faces). (b) The panorama colored by camera cluster. (c) The final panorama.*

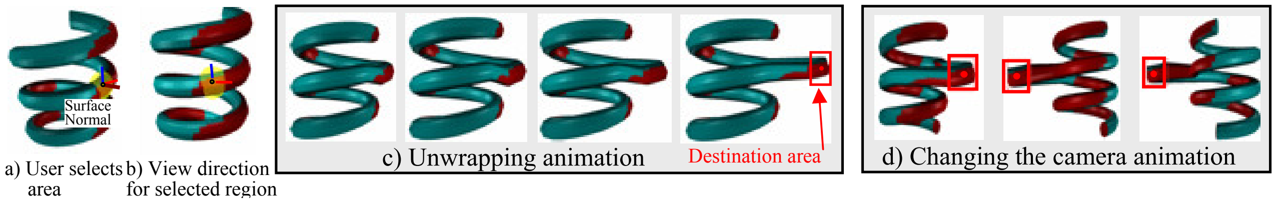

Figure 9: *(a) The user places the unwrap widget on the area of interest to define the source volume. (b) The view direction is automatically set to be the surface normal direction. (c) An animation showing how the selected area is "unwrapped" into the destination area (red box), which is automatically defined. (d) The user changes the global camera. The destination area automatically switches sides as the global camera rotates.*

5 Results and remarks

The unwrap widget is demonstrated in Figures 1, 3, and 9. For the first two figures we show frames of the unwrap animation followed by a rotation of the global camera. The third figure shows the unwrap widget combined with two fish-eye widgets.

The clipping widget is illustrated in Figures 2 and 4. In the first example, a fish-eye widget was added to magnify the contact point of the bones. In the second example we rotate the global camera in order to view both knots from all directions.

We show four panorama images, two of the face (Figure 7), and one each of the cow (Figure 11) and gargoyle (Figure 8). For the face we show one panorama built from nine key frames, the second with only two keyframes that are fairly disparate.

As the number of widgets added into the scene increases, it becomes more difficult to specify how they interact. With a large number of widgets, it also becomes hard to perceive the underlying structure of the data set and the usefulness of the final non-linear projection is reduced. We have found that four independent widgets on a large model (such as the human pelvis) is about all that can reasonably be used.

For the unwrap widget it is an open question about whether or not a distortion, versus two separate views, helps with maintaining spatial context. The unwrap widget attempts to limit the distorted area to parts of the model which are outside of the areas of interest to reduce incorrect spatial cues. Like non-linear magnification in 2D images, the area that is distorted is fairly apparent during animation, i.e., when the tool or view is moving, but less so when the image is still. Providing an effective cue for highlighting where the deformation is and the amount of "bend" [Zanella et al. 2002] might help to reduce confusion.

6 Conclusion

We have presented a flexible, general-purpose framework for building non-linear projections of data sets. Multiple widgets can be combined easily in real-time to allow viewing of several features within a single view. We illustrate the flexibility of our framework by adapting it to create panoramic views.

7 Acknowledgements

The work presented here was funded in part by NSF grant 0238062. The authors would like to thank J. J. Crisco of Brown Medical School, Rhode Island Hospital for the bone data set (captured under NIH AR44005), the AIM Shape repository for the human pelvis model and the gargoyle model and Cyberware for the face scans.

References

AGARWALA, A., AGRAWALA, M., COHEN, M., SALESIN, D., AND SZELISKI, R. 2006. Photographing long scenes with multi-viewpoint panoramas. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM Press, New York, NY, USA, 853–861.

AGRAWALA, M., ZORIN, D., AND MUNZNER, T. 2000. Artistic multiprojection rendering. In *Proceedings of Eurographics Rendering Workshop 2000*, Eurographics, 125–136.

BLINN, J. 1988. Where am I? What am I looking at? In *IEEE Computer Graphics and Applications*, vol. 22, 179–188.

BROSZ, J., SAMAVATI, F. F., SHEELAGH, M. T. C., AND SOUSA, M. C. 2007. Single camera flexible projection. In *NPAR '07: Proceedings of the 5th international symposium on Non-*

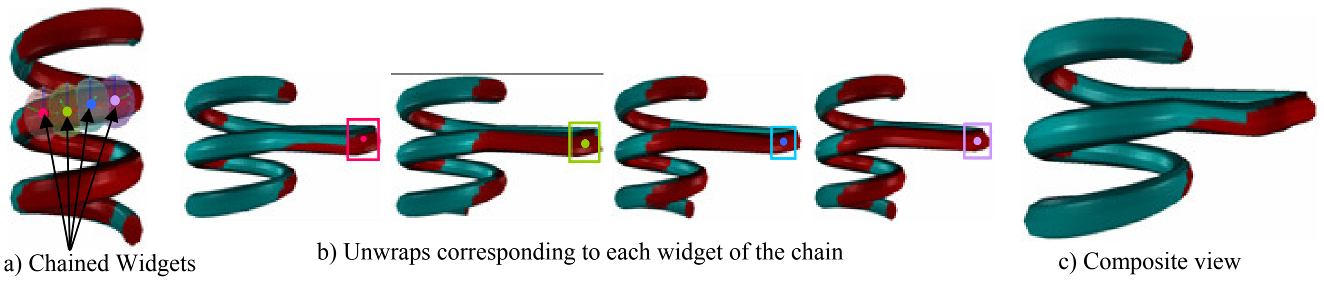

Figure 10: *Our goal is to unwrap a larger portion of one of the coils of the helix (27,689 faces, 13,248 vertices). (a) Adding several instances of the chained unwrap widget along the back of the helix. (b) Each unwrap widget produces a unique unwrapping. (c) These unwraps are combined together to produce the final view. In addition to the local cameras being blended together, the destination positions of the consecutive widgets are shifted as well (Section 4.3).*

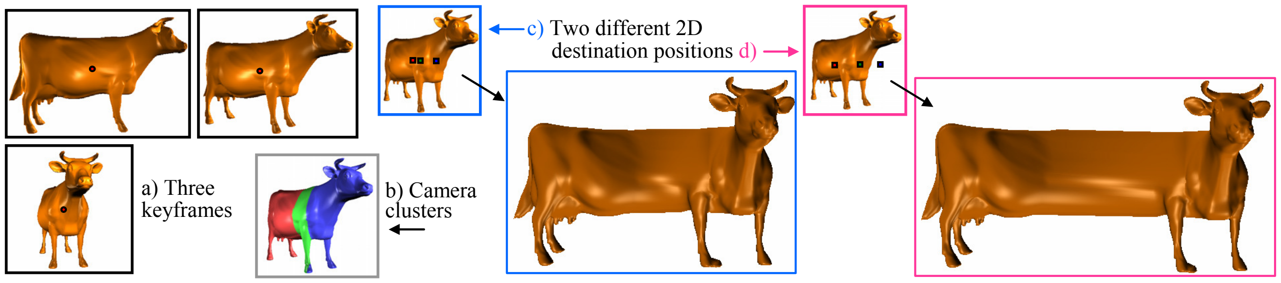

Figure 11: *(a) Three keyframes, along with the centers of the corresponding ROIs, used to generate a stretched-out cow panorama. (b) The cow colored by the camera clusters. (c) The initial destination positions calculated automatically using the method outlined in Section 4.1 along with the corresponding panorama. (d) Modifying the destination positions to stretch the cow out further.*

photorealistic animation and rendering, ACM, New York, NY, USA, 33–42.

BRUCKNER, S., AND GRÖLLER, M. E. 2005. Volumeshop: An interactive system for direct volume illustration. In *Proceedings of IEEE Visualization 2005*, H. R. C. T. Silva, E. Gröller, Ed., 671–678.

CARPENDALE, M. S. T., COWPERTHWAITE, D. J., AND FRACCHIA, F. D. 1997. Extending distortion viewing from 2D to 3D. *IEEE Comput. Graph. Appl. 17*, 4, 42–51.

COLEMAN, P., AND SINGH, K. 2004. Ryan: rendering your animation nonlinearly projected. In *NPAR '04: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, ACM Press, New York, NY, USA, 129–156.

COLEMAN, P., SINGH, K., BARRETT, L., SUDARSANAM, N., AND GRIMM, C. 2005. 3D screen-space widgets for non-linear projection. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, ACM Press, New York, NY, USA, 221–228.

GLASSNER, A. 2004. Digital cubism, part 2. *IEEE Comput. Graph. Appl. 24*, 4, 84–95.

GLEICHER, M., AND WITKIN, A. 1992. Through-the-lens camera control. *Siggraph 26*, 2 (July), 331–340. ISBN 0-201-51585-7. Held in Chicago, Illinois.

GRIMM, C., AND SINGH, K. 2005. Implementing the IBar camera widget. *Journal of Graphics Tools 10*, 3 (November), 51–64. This is the full implementation details for the UIST 2004 paper. There is source code available.

GRIMM, S., BRUCKNER, S., KANITSAR, A., AND GRÖLLER, M. E. 2004. Flexible direct multi-volume rendering in interactive scenes. In *Vision, Modeling, and Visualization (VMV)*, 386–379.

GRIMM, C. 2001. Post-rendering composition for 3D scenes. *Eurographics short papers 20*, 3.

KURZION, Y., AND YAGEL, R. 1997. Interactive space deformation with hardware-assisted rendering. *IEEE Comput. Graph. Appl. 17*, 5, 66–77.

LAMAR, E. C., HAMANN, B., AND JOY, K. I. 2001. A magnification lens for interactive volume visualization. In *Proceedings of Ninth Pacific Conference on Computer Graphics and Applications- Pacific Graphics 2001*, IEEE Computer Society Press, Los Alamitos, California, H. Suzuki, L. Kobbelt, and A. Rockwood, Eds., IEEE, 223–232.

MARTTIN, D., GARCIA, S., AND TORRES, J. C. 2000. Observer dependent deformations in illustration. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, ACM Press, New York, NY, USA, 75–82.

MCGUFFIN, M. J., TANCAU, L., AND BALAKRISHNAN, R. 2003. Using deformations for browsing volumetric data. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, IEEE Computer Society, Washington, DC, USA, 53.

POPESCU, V., MEI, C., DAUBLE, J., AND SACKS, E. 2006. An efficient error-bounded general camera model. In *Proceedings of 3rd International Symposium on Data Processing, Visualization, and Transmission.*

RADEMACHER, P. 1999. View-dependent geometry. In *Siggraph*, ACM Press/Addison-Wesley Publishing Co., ACM, 439–446.

SINGH, K. 2002. A fresh perspective. In *Proceedings of Graphics Interface 2002*, 17–24.

SUDARSANAM, N., GRIMM, C., AND SINGH, K. 2005. Interactive manipulation of projections with a curved perspective. In *Eurographics short papers*, Eurographics, vol. 24 of *Computer Graphics Forum*, 105–108. A specific type of non-linear projection that takes vanishing lines to sinusoids.

SZELISKI, R. 1996. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications 16*, 2, 22–30.

TAKAHASHI, S., OHTA, N., NAKAMURA, H., TAKESHIMA, Y., AND FUJISHIRO, I. 2002. Modeling surperspective projection of landscapes for geographical guide-map generation. *Comput. Graph. Forum 21*, 3.

WANG, L., ZHAO, Y., MUELLER, K., AND KAUFMAN, A. 2005. The magic volume lens: An interactive focus+context technique for volume rendering. In *Proceedings of IEEE Visualization (VIS) 2005*, 367–374.

WOOD, D. N., FINKELSTEIN, A., HUGHES, J. F., THAYER, C. E., AND SALESIN, D. H. 1997. Multiperspective panoramas for cel animation. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., ACM, 243–250.

YANG, Y., CHEN, J. X., AND BEHESHTI, M. 2005. Nonlinear perspective projections and magic lenses: 3D view deformation. *IEEE Comput. Graph. Appl. 25*, 1, 76–84.

ZANELLA, A., CARPENDALE, M. S. T., AND ROUNDING, M. 2002. On the effects of viewing cues in comprehending distortions. In *NordiCHI '02: Proceedings of the second Nordic conference on Human-computer interaction*, ACM, New York, NY, USA, 119–128.