INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

View-Dependent Layered Projective Texture Maps

Alex Reche — George Drettakis

N° 5016

Novembre 2003

THÈME 4

ISSN 0249-0803

View-Dependent Layered Projective Texture Maps

Alex Reche*† , George Drettakis ‡†

Thème 4 — Simulation et optimisation
de systèmes complexes
Projets Reves

Rapport technique n° 5016 — Novembre 2003 — 19 pages

Abstract: Capturing and rendering of real scenes in an immersive virtual environment is still a challenging task. In this paper we present a novel workflow, which emphasizes high-quality, view-dependent projective texturing at low memory cost, while allowing artist intervention to ensure image quality control. First, a small set of photographs of a real scene are used to create a 3D model, using standard tools to create models from images. Optimized visibility layers are then created for each photograph. These layers are subsequently used to created standard 2D image-editing layers, enabling artists to fill-in missing texture using standard techniques such as clone brushing, or to create transparency maps. The result of this preprocess is used by our novel layered projective texture rendering algorithm, which has low texture memory consumption, high interactive image quality and avoids the need for subdividing geometry for visibility. Our approach brings together a set of standard components and tools, and adapts easily to existing workflows, both in terms of content creation and virtual environment display. We show results of our implementation on a real-world project.

Key-words: Virtual Reality, Augmented Reality, Projective Textures, Layered Images, View Dependent Texture Maps

This is a note

This is a second note

* Footnote for first author
† Shared foot note
‡ Footnote for second author

Textures Projectives a Calques dependant du Point de Vue

Résumé : La capture et le rendu des scènes réelles dans le contexte des environnements virtuels immersifs sont encore des tâches très difficiles. Nous présentons une nouvelle approche, en utilisant des textures projectives dépendantes du point de vue, de haute qualité et avec un faible coût en mémoire texture. Dans un premier temps, des photographies de la scène réelle sont utilisées pour créer un modèle 3D avec des outils standards. Notre méthode ordonne automatiquement la géométrie en groupes à niveaux de visibilité différents pour chaque point de vue. Ces groupes sont ensuite utilisés pour découper les images en calques, permettant aux artistes le remplissage des textures manquantes, dues à la non-visibilité, en utilisant des outils comme le tampon de duplication (clone brush). Le résultat de ce processus est utilisé par notre nouvel algorithme de rendu en utilisant des textures projectives à calques, qui a un faible coût d'utilisation de mémoire de texture, une qualité haute de rendu et dans le cas de textures projectives dépendantes du point de vue, évite la subdivision de la géométrie engendrée par la visibilité en utilisant des méthodes précédentes.

Mots-clés : Realite Virtuelle, Realite Augmentee, Textures Projectives, Calques d'Images, Textures dependantes du point de vue

Contents

1 Introduction

Recent developments in modeling from images and other real-world capture techniques (e.g., laser or time-of-flight scanning, or stereo-based methods), have resulted in an increasing interest in the creation and display of 3D models in realistic computer graphics and virtual environments (VE's). The visual quality of such environments can be very high, thanks to the richness of the textures extracted from high-resolution digital photography and the quality of the reconstructed 3D geometry.

The applications for such VE's are numerous. Urban planning and environmental impact studies, archaeology and education, training, design but also film production or computer games are just a few cases where these highly-realistic virtual environments comprise an important quality and technological advancement.

Modeling-from-images approaches have been largely based on pioneering work in Computer Vision [20, 5]. In Computer Graphics, the work by Debevec [3], and his Facade system have inspired much of the work which followed. Image-based modelling and rendering methods have also been developed [12, 10, 6], which are yet another alternative method to capture and display real world scenes. The Facade approach, and the follow-up interactive rendering version [4], use a view-dependent rendering algorithm. The textures used in display are blended from textures from multiple views, and can be quite satisfactory. In the interactive case, projective textures are used, requiring a geometric visibility pre-process (subdivision of polygons).

In practice, several commercial modeling-from-images products have been since developed and are used for real world projects (e.g., [27, 23, 26]). All of these products however have adopted standard, rather than projective, texture mapping for display: each polygon has an associated texture, which is extracted from the input photographs by applying inverse camera projection. Geometry is thus displayed as regular textured polygons, and can be directly integrated into a traditional computer graphics or virtual environment rendering system. This choice is also justified by a need for control of visual quality. Artists can intervene at various stages of the process, using standard image-editing tools (such as Adobe Photoshop^{TM}[7], or GIMP[25]), for example to fill in missing texture information or remove undesirable objects etc. They can also edit the textured geometry using standard modeling tools [22, 24], in a traditional graphics/VE production workflow.

In this paper, we present new algorithms and a new workflow which address a number of shortcomings of previous methods.

- We adopt a view-dependent display model, by blending textures coming from different input photographs corresponding to different viewpoints; the quality of the renderings is thus much higher than the result of standard modeling-from-images products. One goal of our approach is to remain compatible with standard graphics and VE rendering systems, typically in the context of a scene-graph based approach.

- We develop a new projective texture rendering approach, which does not require subdivision of polygons. This approach is based on image-layers, which are automatically

generated by our system. This also reduces texture memory requirements compared to texture extraction approaches.

- The image-layers are tightly integrated with standard image-editing programs. Artists can thus edit the image in a standard manner, maintaining tight control over final image quality.

We have implemented our system, and we show results from a reconstructed 3D model of a real-world project, which is the construction of the Nice Tramway. For comparison, we show how existing commercial products can be used to create view-dependent renderings, and we show by example that our method has lower memory consumption, superior visual quality and is easier to use and faster for model creation.

2 Related Previous Work

We first review the most closely related work which is that of Debevec and colleagues, on view-dependent texture mapping. We briefly mention other image-based modeling and rendering (IBMR) approaches, explaining why they cannot be used for our purposes. Finally we briefly discuss layering, visibility ordering and image editing.

2.1 View Dependent Texture Mapping

In their original paper [3], Debevec et al. presented the first modeling-from-images system Facade. The idea of view-dependent texture mapping (VDTM) was presented, but the rendering approach is clearly off-line and uses model-based stereo to obtain very impressive visual results. This work was transposed to an interactive rendering context [4], in which the polygons which are partially visible in one of the input images are subdivided and missing textures are filled by simple color interpolation. Projective texturing is used and textures are blended by creating a view-map of closest viewing angles. This method has the advantage of being completely automatic; the flip side of this is the lack of quality control in a traditional content-creation workflow. In particular, for surfaces which do not receive a projective texture a hole-filling technique is used. This step, which is one of the main sources of visual error, is done by simple interpolation of colours from neighouring vertices. Subdivision of input geometry could also be problematic in some contexts, due to numerical imprecision. The method we will present can be seen as an extension of this approach, by addressing these two issues.

2.2 Image-based Modeling and Rendering

Image-based modeling and rendering solutions have been developed since the mid-nineties; pioneering work includes view-interpolation [2] and plenoptic modeling [12]. The Lightfield[10] and Lumigraph[6] algorithms are based on relatively dense sampling of images, so that pure

(Lightfield) or geometry-assisted (Lumigraph) image interpolation suffices to render an image. The size of the data sets required for these approaches (gigabytes of image data), makes them very hard to use for realistic projects.

Many improvements have been proposed to these basic methods. Examples include view-based rendering [17] or surface light fields [21]. One method of particular interest is the Unstructured Lumigraph [1], which can be seen as a "middle ground" between the lumigraph and view-dependent texture mapping. The criteria for blending are more sophisticated compared to VDTM, and the geometric model can be almost arbitrarily coarse. All of these methods require special-purpose rendering algorithms, and is some cases are hard to integrate in a traditional scene-graph like rendering structures, in which captured and "standard" virtual objects co-exist, and can be manipulated interactively.

Another IBMR approach are layered depth images[18], which add depth to a multi-layer image, resulting in high-quality renderings. Both real (simple layers) and synthetic examples were presented. Special-purpose algorithms are also necessary for this approach. A related method is relief texture mapping[15], which renders geometric detail using a depth map, and performs part of the projective transformation by manipulating the texture plus depth map before rendering. The results, both for Relief Texture Mapping and for LDI's, are dependent on the quality and availability of depth information.

2.3 Visibility Ordering and Image-Editing

Our algorithm for layer construction is based on a hardware visibility ordering algorithm by Krishnan et al.[9]. A more complete, software solution has been presented by Snyder and Lengyel [19]. Visibility cycles are resolved in the algorithm by Newell et al. [13]. Our work is also related to the work of Oh et al.[14], in what concerns the integration of 3D information and image editing. Most of the algorithms developed by Oh et al. could be applied to our image editing operations, making them more efficient and easy to use.

3 Overview

We will first discuss 3D model reconstruction using modeling-from-images solutions, and how they can be adapted to view-dependent rendering.

We will then describe our image layer construction algorithm. Our goal is to achieve projective texture mapping, without the need to subdivide input geometry. Consider the example shown in Fig. 1. The scene has three objects, A, B and C and we consider 3 views, cameras c_1, c_2 and c_3. The corresponding images are shown from each camera in the second row. When using the previous projective texture mapping algorithm[4], even for such a simple scene, objects A and B must be subdivided. For view c_1, B will be cut into the B_1, B_2 (see Fig. 1 top row). When moving from c_1 to c_2, object B_2 will have the image from c_1 as a projective texture, and then will blend the images from c_1 and c_2. Object B_1 however, will always be displayed with the image of c_2, since it is invisible from c_1.

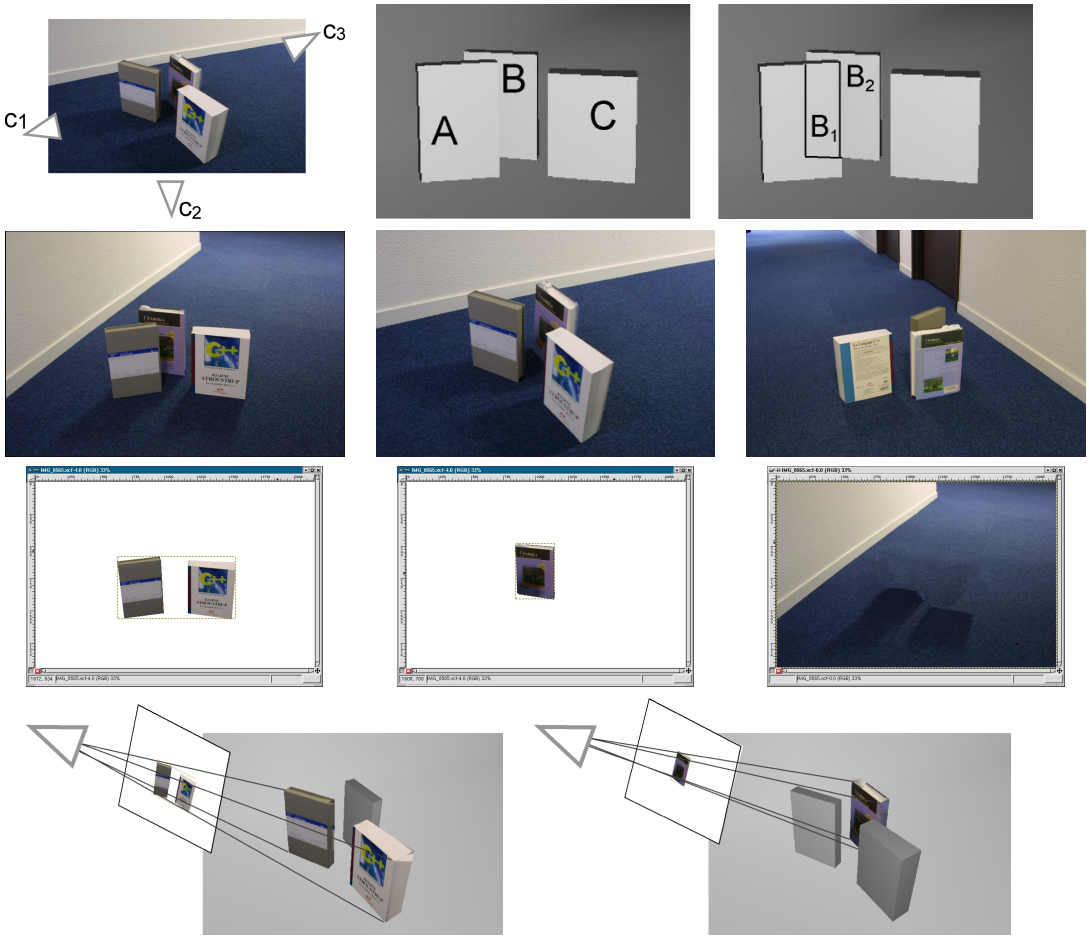

Figure 1: Row 1 left: The geometric configuration of the scenes and the cameras. Row 1 right: the corresponding 3D geometry. Row 2: The three images corresponding to the three cameras, c_1, c_2 and c_3. Row 3: The three image-layers viewed in GIMP, corresponding to camera c_1, after image editing (clone brushing etc.). Row 4: Visualisation of the projected layers.

Instead of doing this, we create layered images, or *image layers*, by first constructing *visibility layers*. For a given camera or view, a geometry layer is a set of surfaces for which no pair is mutually occluded. In the example of Fig. 1, and for camera c_1, there are three layers, the first containing A and C, the second containing B, which is partially hidden by A and the third the background We then create an *image layer*, corresponding to each

geometry layer (see Fig. 1 third row). The first layer corresponds to the portions of the image from c_1 covered by the pixels corresponding to A and C, and the second image layer corresponds to object B.

We construct geometric visibility layers by adapting the hardware-based algorithm of [9], and then create standard image-editing (i.e., Adobe Photoshop[7] or GIMP[25]) layers, which will be used as projective textures. We discuss how these layers can be edited using standard techniques. For example, for the layers of c_1, the missing texture of B is filled in GIMP.

The edited layers are then used as projective textures in our new algorithm. The algorithm uses the corresponding compact image layer as projective textures for the appropriate polygons of the reconstructed 3D model, avoiding the need for geometry subdivision. This is shown in the last row of Fig. 1. Because the layers and image-editing have resolved visibility we can move around freely.

We have implemented this entire workflow, using REALVIZ ImageModeler[27] to reconstruct 3D geometry, and GIMP[25] as an image-editing tool. We present the method and results on a model of Place Massena in Nice, showing how we can rapidly construct appropriate texture layers, and achieve interactive, high-quality, view-dependent display of captured real scenes.

4 Real Scene Capture

The first step of our approach is to create the coarse 3D model; we have chosen to use an approach based on modeling from images. We first calibrate the cameras of the input photographs, and we construct an approximate, polygonal 3D model of the real scene. In this work we use REALVIZ ImageModeler^{TM}[27]. Other products such as PhotoModeler[26] or Canoma [23], or research systems such as Rekon[16] could be used with the same result. The research system Facade[3, 4], can also be used to the same effect, but has a different, view-dependent approach to rendering, as described in the Sect. 2.

These systems [27, 23, 26, 16] extract textures (see Fig. 2), either from a single image or from a combination of images. Texture extraction results in an "unfolding" of the corresponding part of the input photograph into texture space, by applying the inverse perspective transform. The texture can then be applied via standard texture mapping onto the 3D model generated.

As a result, the overall texture requirements using this method are much higher than the resolution of the input images. Consider for example Fig. 2, where the entire input photograph has resolution 2160 x 1440 and just the facade texture has resolution 2323 x 439.

Textures extracted by this process are not always complete. Consider for example the last row of 3; on the left we see the result of extracted textures before editing, and on the right we see the edited result. The final result is obtained using standard image-editing programs such as Adobe Photoshop^{TM}[7] or GIMP[25]. An artist will typically use clone-brushing or simple editing to repair this problems. Extracted textured are "unfolded" using the inverse

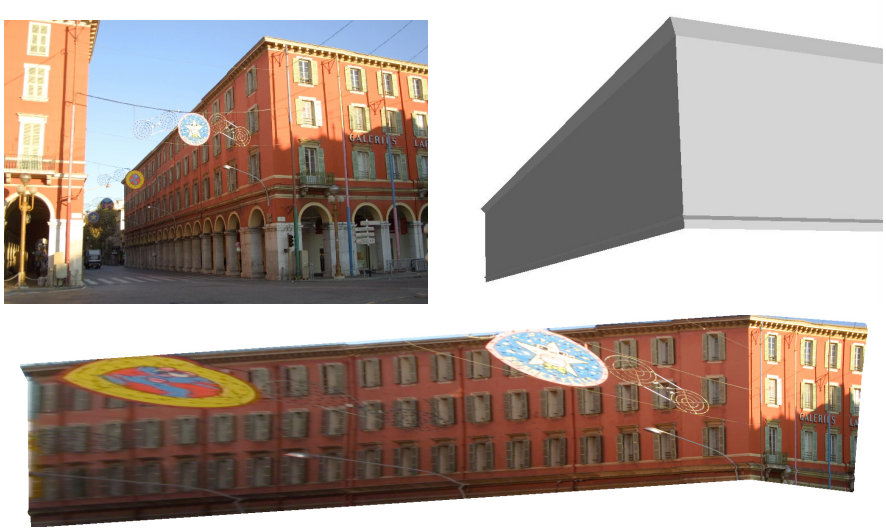

Figure 2: Top: the photo of a facade and the corresponding 3D model. Bottom: the extracted texture. Notice how the windows at the extremities have been distorted by perspective unfolding.

perspective transfrom and editing is typically performed in this space. Although unfolded texture intuitively correspond to object space, and can be convenient for some editing tasks, artists are used to clone-brushing directly in image space, and can find directly editing the image more convenient. Unfolding textures is fundamentally a inverse projection and resampling problem, and thus is very sensitive to the algorithms used. The unfolded texture can have insufficient resolution in certain regions, and its shape can be unintuitive if a texture is assigned to compound objects (see Fig. 2).

View-dependent viewing can also be effected, by displaying the reconstructed model using multiple textures, one per view. We have implemented this and used it in a virtual/augmented reality system [11]; however the construction and editing of the scenes is very cumbersome and the requirements in texture memory extremely high.

5 Creating Visibility and Image Layers

We now describe our new image layering approach. The first step is to create a set of images with calibrated cameras, and a corresponding coarse 3D model of the scene (Fig. 3). Our goal is to create a set of layers for each input image of the set, which will be subsequently used as projective textures for display. These layers have the following property: for any object in the first layer, no other object in the scene occludes it; for the second layer, once

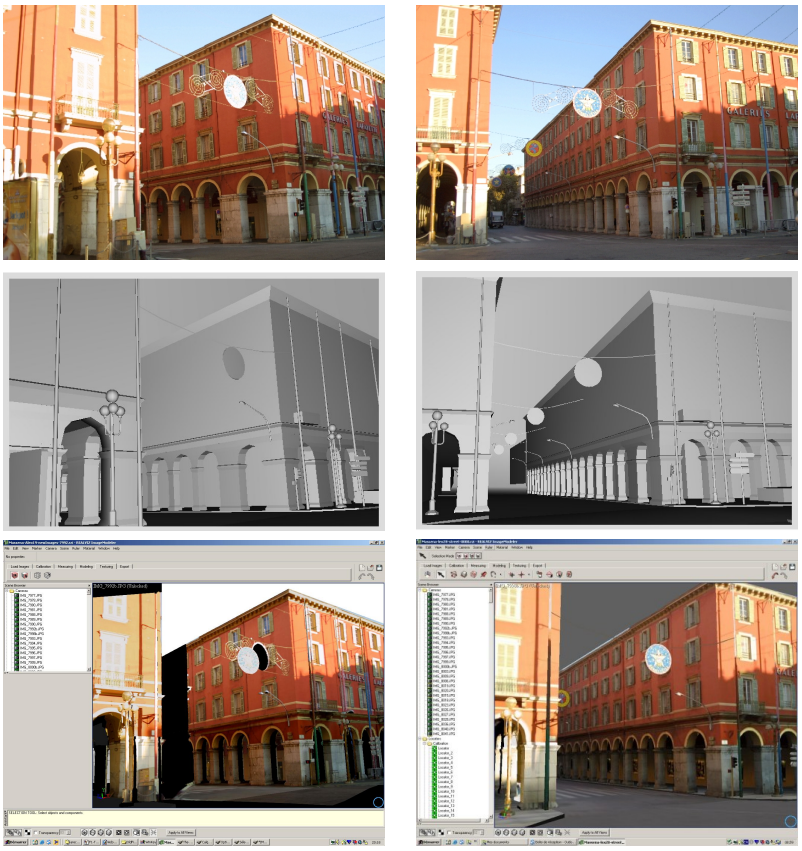

Figure 3: First row, 2 of the 3 input photographs used in this example. Second row, the 3D model extracted, viewed from the same viewpoints (snapshots from ImageModeler^{TM}. Third row, one intermediate views before texture editing and one after texture editing.

the objects of the first layer have been removed, the same property applies, and so on for each layer. Once these layers have been created, we can use projective texture mapping without the need to subdivide geometry, since we create a separate projective texture map for each layer.

As mentioned previously, we tightly integrate the creation of these layers with a standard image editing program, so that an artist can fill in the missing parts of a given layer use standard techniques such as clone brushing etc.

To create these image layers, we first perform a visibility sort in 3D, to create what we call *visibility layers*, which are groups of polygons. We then create what we call *image layers*,

which are true image-editing-programme layers, usable in an image-editing programme such as Adobe Photoshop^{TM}[7] or GIMP[25].

For the first step, we adapt an existing visibility sorting algorithm and then optimize the layers so that we do not waste image resolution. The second step is achieved by sending the output of the visibility sorting algorithm to the editing program which then creates standard image-editing 2D layers, so that an artist can subsequently edit the result.

5.1 Visibility Layers

To compute the visibility layers we adapt the visibility algorithm of Krishnan [9]. Initially, we render the entire scene into an item buffer, i.e., we use a unique identifier for each object in the scene. We then read this buffer to create an initial working set of potentially visible polygons, which thus contains all objects with at least one visible item buffer pixel containing its id.

The algorithm then iteratively constructs a set of visibility layers. For each iteration, the item-buffer of the working set is rendered into the color and the stencil buffer, and the z-test inverted (`GL_LESS`). For each pixel with a stencil value greater than 1, the object corresponding to the item-buffer value of the pixel is partially hidden, and thus the object is removed from the working set. The working set is then rendered iteratively, until the stencil buffer contains values less than or equal to 1. The remaining objects are those completely visible, so they will be the objects for the current layer. However, an additional test is done to the set of objects to solve a problem in the algorithm of Krishnan [9]. We can see an illustration of this problem in Fig. 4. In this case the algorithm chooses objects A and C as objects for the first layer. However, the only completely visible object is object A. The additional test consists in a comparison between the first normal rendering pass and an additional normal rendering pass with all the selected objects. All objects that are visible in the final pass and not completely visible in the first pass, cannot be considered as completely visible. Thus, they are removed from the current working set. All the objects selected are then inserted into the current visibility layer, and removed from the working set. The current layer is saved, then set to empty, and the iteration continues.

The output of this algorithm is a set of visibility layers. In each visibility layer we have a set of polygons, which are used to build layered images.

In our implementation we have optimized this process hierarchically thanks to the structure of the scene which is provided by the image modelling programme. In particular, the scene provided by ImageModeler is organised into objects which contain a set of faces or polygons. Initially, identifiers are given to objects rather than faces, and the algorithm is performed on objects, rapidly eliminating large objects. When we can no longer proceed with objects, we split them into their constituent polygons. In our examples this is enough to resolve the visibility in the scene. Nonetheless, cycles can occur, even at the level of individual polygons/triangles, and in this case we would need to identify the cycles and cut the geometry using an algorithm such as that of Newell et al. [13].

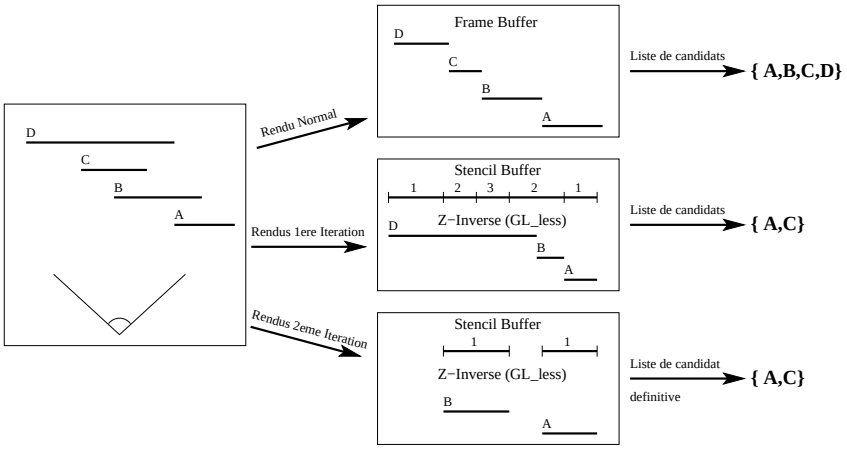

Figure 4: Illustration of the problem in the original [9] algorithm, and our solution.

5.2 Creating Image Layers

To create the image layers, we start with the set of visibility layers. For each visibility layer, we project each constituent polygon into image space, resulting in a 2D contour for each. We then perform a simple clustering algorithm for the set of contours. We take advantage of the fact that due to `OpenGL` restrictions, each texture has to have resolution of a power of 2. For a pair of contours c_1 and c_2 we merge c_1 and c_2 into the same layer, if:

$$A_{12} < A_1 + A_2, \tag{1}$$

where A_i is the minimal bounding box with resolution in x and y powers of two, with area greater than the area of the bounding box of the projected contour c_i. Thus each layer is potentially split into several layers. Examples of image layers can be seen in Fig. 1 and Fig. 5.

This results in an increase of the number of layers, but is required to reduce the texture memory requirements of our projective textures. This is because very distant objects can belong to the same visibility layer, which would result in textures with large areas of empty space, wasting texture memory. The final set of layers is then ready for image editing in the appropriate program. These layers consist of a 2D bounding box in image space for the image being treated. This process is shown in Fig. 5.

The layers are then sent to the image-editing program (GIMP [25] in our case), which creates a 2D image layer corresponding to the input image. An example of such layers is shown in Fig. 5.

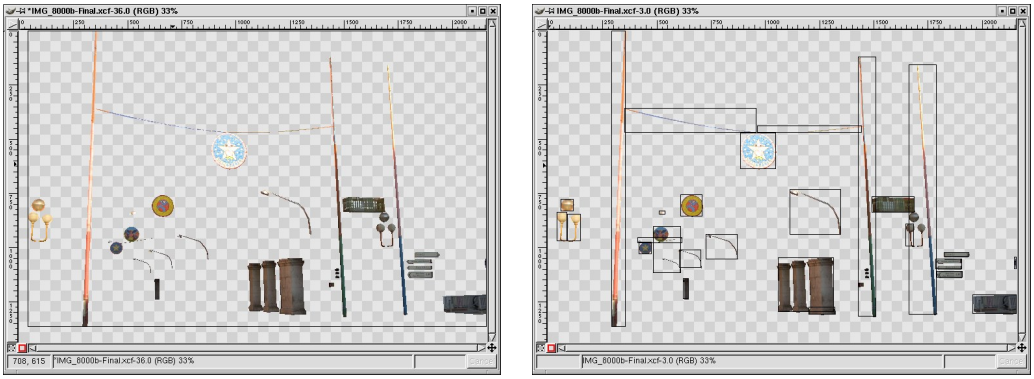

Figure 5: Left: A visibility layer before optimisation. All objects shown are in the same layer. Right: The resulting layer after optimisation to avoid empty spaces, and reduce overall texture memory.

6 Image Editing

Once layers have been created in the editing program, an artist can perform standard operations required to fill in missing details, or to create transparency maps, which are particularly effective for multiple-view rendering.

We return to the example of the facade cited earlier (Fig. 2). Using our new approach, the artist now can use all standard image-editing, such as clone brushing in the usual manner, directly in image space.

In Fig. 6, we see the layers corresponding to the facade and to the posts in front. The artist works in the layers of the facade, performs clone-brushing and the work is complete.

This approach is particularly useful for small details. Perspective correction in the style of [14] could be easily added in this case and would render the tool even more useful and efficient.

Another useful kind of edit which is practical in this context is the manual creation of alpha maps. Again, artists are used to working in perspective image space, and thus creating alpha-maps is easier.

7 Layered View Dependent Projective Texture Display

After creating the visibility layers and manually editing the texture layers for the objects in the scene we can now use projective texture mapping to display the captured real scene.

For any viewpoint corresponding to the camera of a photograph, all we need to do is to render the geometry, and for each object assign as a texture the corresponding layer. To

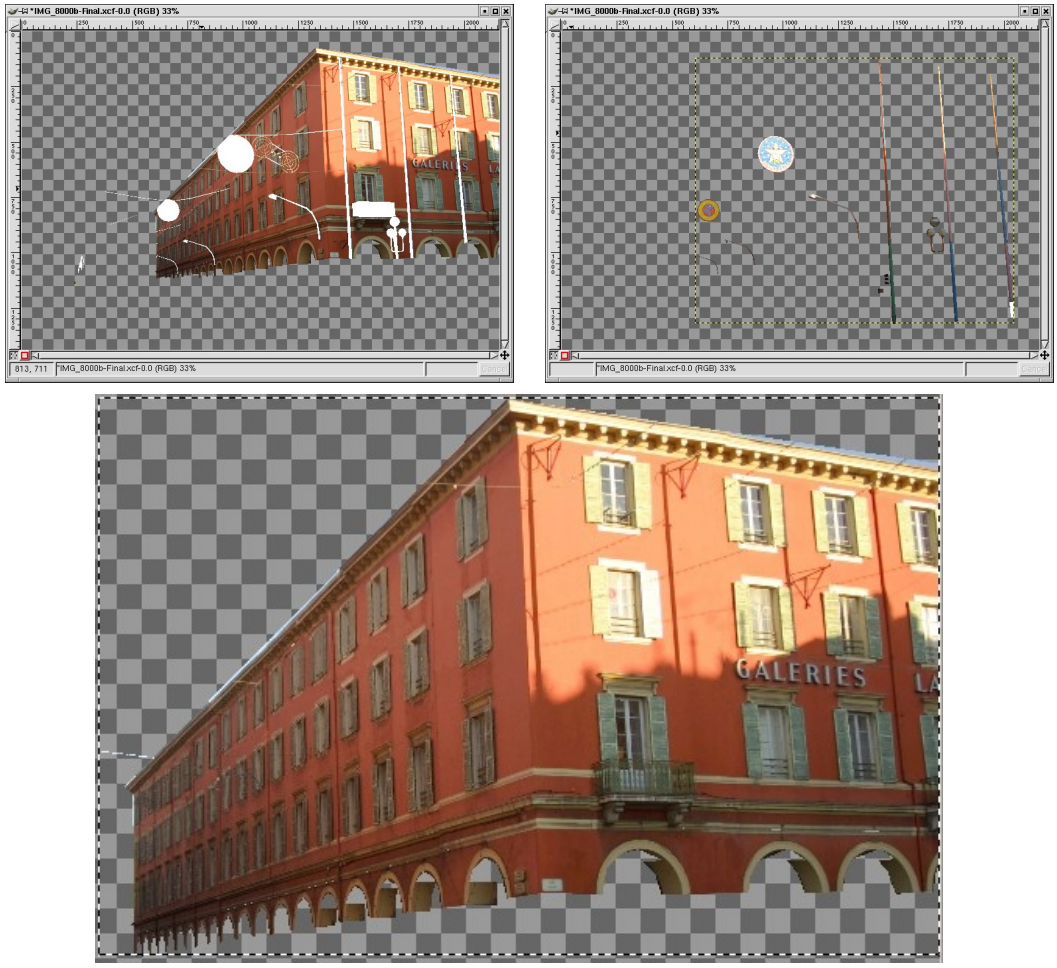

Figure 6: Top row: the layers of the facade and the posts in front. Bottom row: the result after clone brushing on the facade.

allow viewing from other viewpoints, we will blend between textures in the spirit of [4]. We explain this process in detail in what follows.

7.1 Data Structures for Layered View Dependent Projective Textures

As mentioned previously we have designed our approach to fit into a typical scene-graph based rendering system such as OpenGL Performer^{TM}[8]. We thus describe the data structures in terms of scene graph nodes and traversals.

For standard projective texture mapping, a scene-graph based implementation can have the form shown in Fig. 7(a). An image texture node is created, and associated to the geometry nodes (typically polygons). The texture node contains the texture, its width and height and the texture projection matrix (i.e., position of the camera, view direction, up vector, field-of-view near and far).

For layered projective texture mapping, the graph is modified to contain a sub-image texture node, which in addition to the information of a "standard node" described above, contains the image-coordinates of the sub-image (x_i, y_i, w_i, h_i) Fig. 7(b).

To render the layers, we need to apply a sheared perspective, transformation, shown in Fig. 8. This can be directly implemented using the OpenGL command `glFrustum` with parameters:

$$
\begin{aligned}
x'_{min} &= x_{min} + \frac{x_i(x_{max}-x_{min})}{w} \\
x'_{max} &= x_{min} + \frac{(x_i+w_i)(x_{max}-x_{min})}{w} \\
y'_{min} &= y_{min} + \frac{y_i(y_{may}-y_{min})}{w} \\
y'_{may} &= y_{min} + \frac{(y_i+w_i)(y_{may}-y_{min})}{w}
\end{aligned}
$$

7.2 Rendering Multiple Views

To render multiple views, we need to further modify our scene graph structure. We create a view-dependent projective texture node VDP which contains a list of view-dependent texture nodes, associated with each (unique) geometry.

When encountering such a node during graph traversal, the renderer chooses the blending factors for each appropriate texture, sets up the texture matrix for the nodes with non-zero blending factors, and renders the associated geometry in as many passes are required.

8 Results

We show here results of our system; our submission is accompanied by an electronic movie with captures of an interactive session of our system. We use the same example as that presented in previous sections, which is a reconstructed model of Place Massena in Nice. This capture has been performed in the context of a virtual/augmented environment simulation of the Tramway project in Nice. The Tramway will pass between the building shown here and a station will be placed at this location.

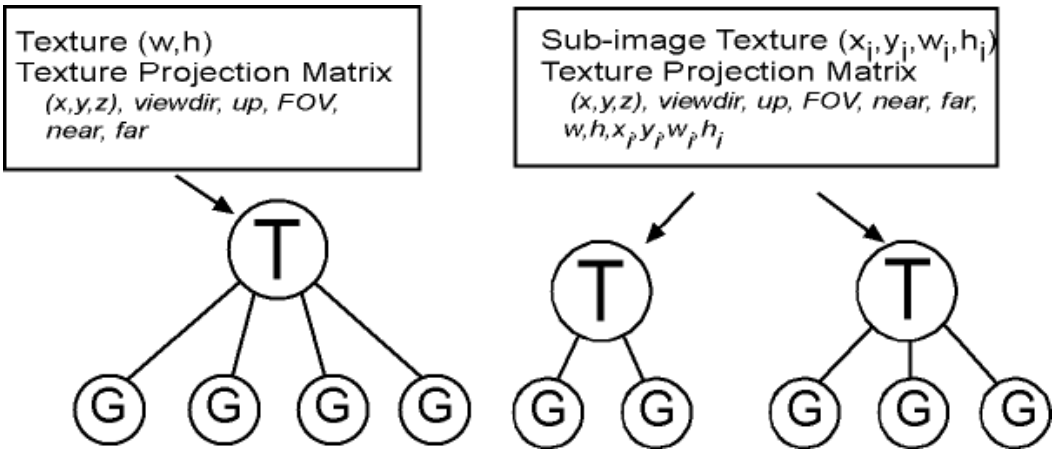

Figure 7: Left: scene-graph implementation of standard projective texturing. Right: scene-graph implementation of layered projective texturing.

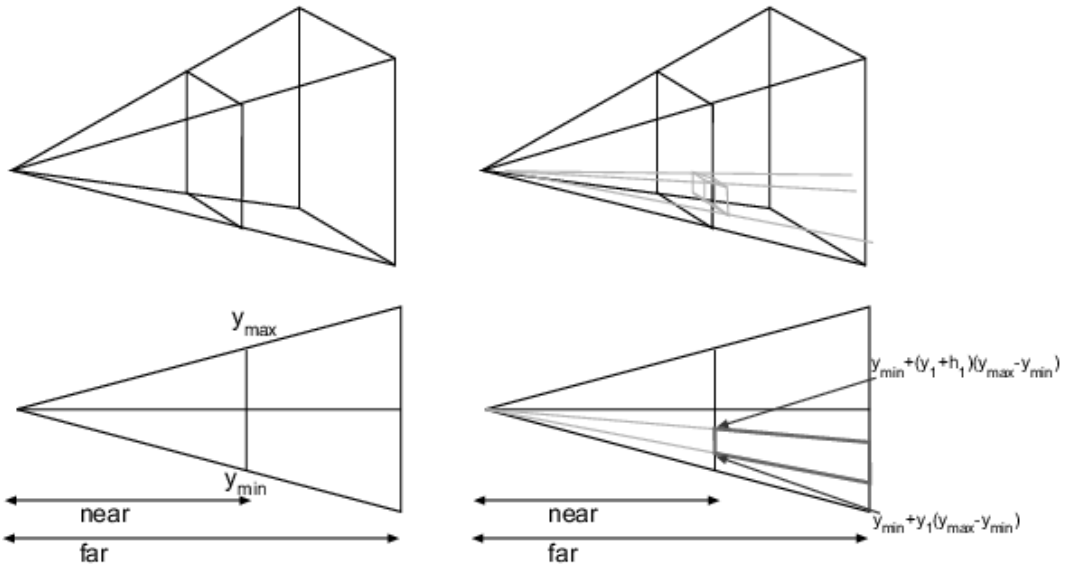

Figure 8: Left: Standard perspective frustum, Right: sheared frustum for use with layers.

In Fig. 9, we show intermediate views (i.e., different from the input photos of resolution 2160x1440) using our view-dependent projective texture method. The algorithm creates a

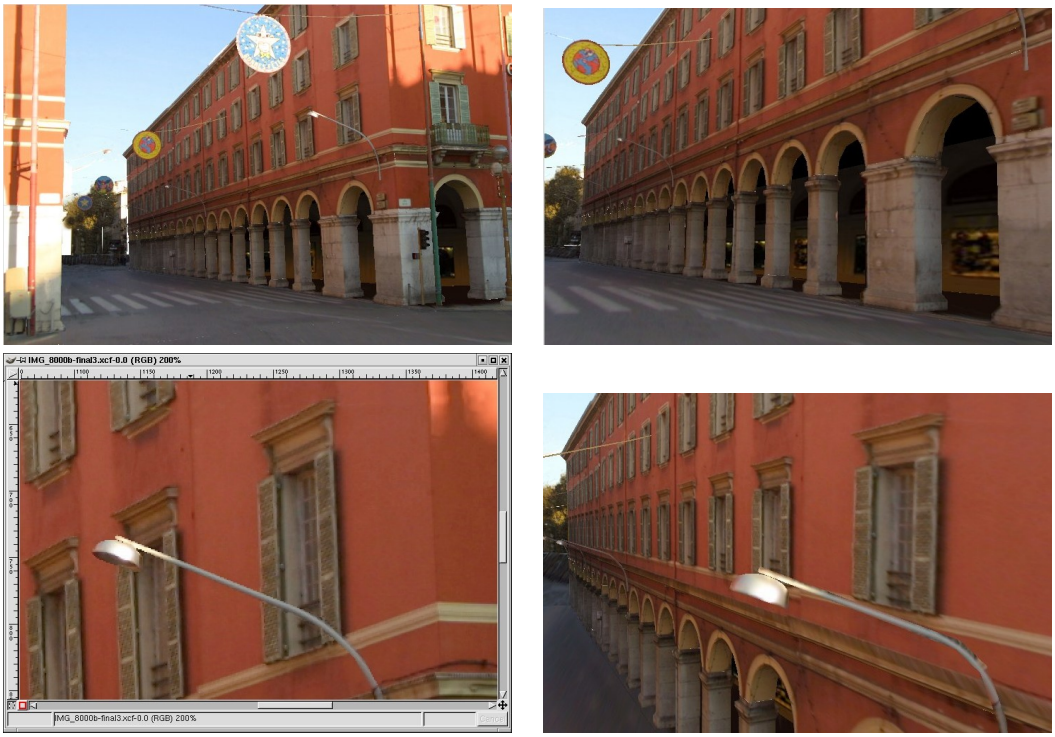

Figure 9: Top row: two intermediate (non-input) views, using our new, layered projective texture display. Lower row (right): Closeup of input image; (left) closeup of rendering of our system from a different viewpoint.

total of between 98 and 160 (optimized) layers for each image, many of which are very small and require no editing. The total texture memory required is between 10-12 Megabytes, depending on the image. The algorithm to create the layers takes less than 3 minutes to complete (most of which is spent in interprocess communication between GIMP and our system). The entire process of editing and cloning the layers took about two days; using a texture extraction-based approach, with the same scene and the same views required three weeks.

As we can see, the quality of the images, and in particular the zoomed-in images is better using our approach, since textures have not been altered with inverse perspective distortion.

9 Discussion and Conclusions

We have presented a new workflow for displaying captured real scenes, using projective texture mapping. We create image-layers of the input photographs, which can be edited in a standard image-editing program to fill in missing texture. The layers are then used as projective textures of the geometry. This is done for a small number of multiple views, and the rendering algorithm blends between the closest views to the current viewpoint.

Compared to the initial interactive projective texture mapping approach [4], we avoid the need to subdivide geometry for visibility, and we introduce more traditional artist intervention to manually fill in missing textures, rather than depending on interpolation. Thus the advantages of our approach are that we avoid increasing the polygon count of the scene, and that the approach fits well with traditional workflows, where artist's control over image quality is paramount; the disadvantage is that hole-filling is no longer automatic.

Compared to texture-extraction methods, our approach significantly reduces texture memory requirements and results in better texture quality, since inverse perspective results in large textures and quality loss due to resampling.

In addition, for some image-editing tasks, artists are used to working directly in image-space, rather than in inverse perspective space, and thus the image-editing process is faster overall.

The high image quality very important. Informal observation of our system in use has shown that the user is much less sensitive to parallax errors due to the high quality of the projective textures used, and thus a much smaller number of input photographs is required, compared to previous methods.

This is just the first step in building complete and usable solutions to effective, high-quality renderings of captured real scenes for immersive systems.

On the capture side, there are many preprocessing tasks which require care and manual intervention, such as colour balancing between multiple views, or ensuring that shadows are in the right place due to motion of the sun. We are investigating ways to make these tasks automatic or semi automatic. Similarly, given that we have multiple views, texture synthesis approaches could be used to avoid manual clone brushing to some extent.

For rendering, we will be integrating this approach into our immersive virtual environment system and we will be investigating how many views are required using an experimental approach.

10 Acknowledgements

The first author is funded by a CIFRE doctoral fellowship, in partnership with the Centre Scientifique et Technique du Batiment (http://www.cstb.fr). This research was partially funded by the EU IST project CREATE, IST-2001-34231, http://www.cs.ucl.ac.uk/create. ImageModeler was gracefully provided by REALVIZ in the context of the CREATE project. Thanks to our lab artist Alexandre Olivier-Mangon for his helpful comments and for editing of part of the example images. Thanks to Frédo Durand for reading and commenting on an early draft.

References

[1] C. Buehler et al. Unstructured lumigraph rendering. In *SIGGRAPH 2001, Computer Graphics Proc.*, Annual Conference Series, pages 425–432, 2001.

[2] S. E. Chen and L. Williams. View interpolation for image synthesis. *Computer Graphics*, 27(Annual Conference Series):279–288, 1993.

[3] P. Debevec et al. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proc. SIGGRAPH 96*, pages 11–20, August 1996.

[4] P. Debevec et al. Efficient view-dependent image-based rendering with projective texture-mapping. In *Rendering Techniques '98*, 9th EG workshop on Rendering, Vienna, Austria, June 1998. Springer Verlag.

[5] O. Faugeras et al. 3-d reconstruction of urban scenes from image sequences. *CVGIP: Image Understanding*, 1997.

[6] S. J. Gortler et al. The lumigraph. In *SIGGRAPH 96 Conference Proc.*, Annual Conference Series, pages 43–54, Aug. 1996.

[7] http://www.adobe.com/products/photoshop/main.html.

[8] P. S. L. http://www.sgi.com/software/performer/.

[9] S. Krishnan et al. A Hardware-Assisted Visibility-Ordering algorithm with applications to volume rendering. In *Data Visualization 2001*, pages 233–242, 2001.

[10] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH 96 Conference Proc.*, Annual Conference Series, pages 31–42, Aug. 1996.

[11] C. Loscos et al. The CREATE project: Mixed reality for design, education, and cultural heritage with a constructivist approach. In *Proc. of ISMAR 2003 (Poster)*, 2003.

[12] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. *Computer Graphics*, 29(Annual Conference Series):39–46, 1995.

[13] M. E. Newell et al. A solution to the hidden surface problem. In *Proc. of the ACM Nat. Conf.*, pages 443–450, 1972.

[14] B. M. Oh et al. Image-based modeling and photo editing. In *SIGGRAPH 2001, Computer Graphics Proc.*, Annual Conference Series, 2001.

[15] M. M. Oliveira et al. Relief texture mapping. In *Proc. of SIGGRAPH 2000 (New Orleans, La)*, July 2000.

[16] P. Poulin et al. Interactively modeling with photogrammetry. In *Eurographics Rendering Workshop 1998*, pages 93–104, June 1998.

[17] K. Pulli et al. View-based rendering: Visualizing real objects from scanned range and color data. In *Rendering Techniques '97 (Proc. of the 8th EG Workshop on Rendering) held in St. Etienne, France*, pages 23–34, 1997.

[18] J. W. Shade et al. Layered depth images. In *SIGGRAPH 98 Conference Proc.*, volume 32 of *Annual Conference Series*, pages 231–242, 1998.

[19] J. Snyder and J. Lengyel. Visibility sorting and compositing without splitting for image layer decompositions. In *SIGGRAPH 98 Conference Proc.*, 1998.

[20] C. J. Taylor and D. J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Trans. on Pat. Analysis and Mach. Intelligence*, 17(11):1021–1032, 1995.

[21] D. N. Wood et al. Surface light fields for 3D photography. In *SIGGRAPH 2000, Annual Conference Proc.*, pages 287–296, 2000.

[22] www.aliaswavefront.com.

[23] www.canoma.com.

[24] www.discreet.com/products/3dsmax/m.

[25] www.gimp.org.

[26] www.photomodeler.com.

[27] www.realviz.com.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803