

Real-Time Video Painting for a Large-Scale Environment

Jinhui Hu

University of Southern California
3737 Watt Way, PHE 404

USA(90089), Los Angeles, California

jinhuihu@graphics.usc.edu

Suya You

University of Southern California
3737 Watt Way, PHE 404

USA(90089), Los Angeles, California

suyay@graphics.usc.edu

Ulrich Neumann

University of Southern California
3737 Watt Way, PHE 404

USA(90089), Los Angeles, California

uneumann@graphics.usc.edu

ABSTRACT

With the rapid development of modeling and remote sensing technologies, it becomes increasingly more feasible to model a large-scale environment. However, the acquisition of static textures for such a large-scale environment is still a challenging task, often demanding tedious and time-consuming manual interactions. We present a system for real-time video painting, which not only acquires textures automatically from multiple images or video sequences, but also updates the texture data in real time to capture the most up-to-date imagery of the environment. Experiments presented in the paper show that we can create and update textures for a university-campus size environment in real time.

Categories and Subject Descriptors

I.3.3 [Computer Graphics]: Picture/Image Generation – *display algorithm*; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – *Color, shading, shadowing, and texture; Virtual reality*.

General Terms

Algorithms, Performance, Design, Experimentation.

Keywords

Video Painting, Image Registration, Hardware.

1. INTRODUCTION

Creating a large-scale photorealistic environment is important for many virtual reality applications. With the rapid development of technologies for modeling and remote sensing, it becomes increasingly more feasible to model a large-scale environment. In rendering a virtual environment, texture mapping is often employed to produce a realistic image without the tedium of modeling small-scale 3D structures and surface reflectance parameters. However, the acquisition or update of static textures for a large-scale environment is a challenging task that often demands tedious and time-consuming manual interactions.

Efficient generation of textures has become an important research issue in computer graphics and image processing. There are two main approaches to creating scene textures, texture synthesis and

direct texture mapping from imagery. Texture synthesis is inspired by research in texture analysis and statistics. While these stochastic textures are useful, this approach is unable to quickly produce complex building and scene textures that are photo-realistic.

Accurate and realistic appearance is feasible when real world images are captured and used as texture-maps. While most graphics systems support high-quality texture mapping, they are optimized for static textures whose mapping to model geometry is specified prior to use. Static textures are usually derived from fixed cameras at known or computed transformations relative to the modeled objects. The creation and management of such texture databases is time-consuming since it includes the creation of mapping functions for each segment of image and corresponding model patch. Therefore, static texture-maps have been difficult to employ in applications that require rapid and dynamic updates of environment imagery.

This paper presents a novel technique, called *real-time video painting*, to cope with the aforementioned limitations of static texture mapping. Live video is used as a texture source and the mapping onto 3D models is computed dynamically, allowing the visualization to reflect the most recent changes in the environments. The video streams can be acquired from stationary or moving cameras, e.g., a handheld camcorder, and the texture mapping onto a 3D model is computed in real time. Unlike the traditional texture mapping process, in which regions of each texture image are *a priori* associated with patches of the geometric model, our approach dynamically creates the associations between the model and texture image as a result of image projection during the rendering process. This allows our method to automate the texture mapping process and update the textures in real time. These capabilities are not feasible with the traditional texture mapping method.

Related Work

A popular texture generation method is texture synthesis [1], which can be further divided into three classes. The first one is a parametric model-based technique, which uses a number of parameters to describe a variety of textures [6] [16]. The second class is non-parametric textures, or example-based methods. These methods generate textures by directly copying pixels from input textures [4]. The third class synthesizes textures by copying whole patches from input images [5]. Texture synthesis has been shown to be a powerful tool that is widely used in many fields. However, the resulting textures are not representative of an actual scene, which limits their application for reconstructing realistic images of real scenes.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Another class of techniques is manually painted textures, which allow users to paint textures. A number of interactive texture painting systems have been presented. Igarashi [8] presents a system that allows a user to directly paint on a 3D model without predefining a UV-mapping. Berman [2] uses Haar wavelet decompositions of images for multi-resolution texture painting. Multi-scale procedural textures are painted on 3D models [15]. Allowing a user to directly paint textures on 3D models makes texture creation and editing much easier for the trained artist. However, as with texture synthesis, the result is not necessarily faithful to a real scene.

A straightforward way to generate realistic textures is to use images as the texture source [9][20]. Multiple static images are often used to create a complete texture map that covers an entire scene model. Bernardini [3] uses high-resolution images to create high quality textures. Rocchini [17] blends multiple images into final textures. Ofek [13] uses quadtree to represent multi-resolution textures extracted from image sequences. However, this method requires the user to mark an area that will be extracted as a texture, and then manually track the area for a short sequence (fewer than 16 images).

Recent work by Schodl [18] treats the repeated patterns in video clips as a video texture. From a finite video clip, the method can generate infinitely long sequences by rearranging and blending the original sequence frames. Soatto et al. [19] present dynamic textures, which are sequences of images with a certain stationary property in time. By learning a model from the input sequence, the method can synthesize new dynamic textures. Both methods use the concept of texture in a non-traditional way.

2. VIDEO PAINTING OVERVIEW

Textures for a large-scale environment can be tedious to acquire through static images, and static images are cumbersome as a means to update scene textures. We pursue the use of live video sequences as texture sources because such sequences are likely to contain overlap between images (no gaps) and represent the most up-to-date scene texture available. However, the use of video also presents several technical challenges that needed to be overcome. A major challenge is the management of potentially infinite texture data since video streams are captured continuously. Other challenges include 3D pose tracking for moving cameras, texture segmentation, and mapping to models.

Our approach to the infinite storage problem is based on the observation that although live video textures present boundless data, 3D models have a well-defined and bounded geometric surface area. We define a *base texture buffer* of specified resolution that is associated with a polygon model or a group of neighboring model-polygons (Figure 2). The base texture buffer is initialized as blank. As video images are processed, a base texture is updated through a warp transformation for any polygon regions that are visible in the image. The update of base texture buffers solves the infinite texture storage problem and avoids the need for polygon clipping in every video frame.

Figure 1 shows the data flow in our video painting system. We briefly describe each component in this section, and further details are provided in [7]. We then present the new contribution of this paper in Section 3, Video Painting for a Large-Scale Environment based on dynamic base-buffer allocation, active buffer

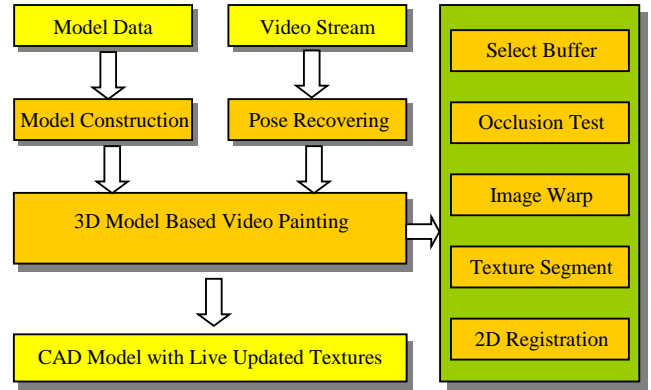


Figure 1. Data flow in the video painting system.

selection, and an efficient implementation that uses graphics hardware.

- **Model Construction and Pose Recovering**

Our 3D model is generated using LiDAR (Light Detection and Ranging) data, and refined based on a semi-automatic approach [22]. We recover the camera pose using a robust tracking approach for image alignment [12]. The pose is further refined during the video painting process to achieve seamless image alignment.

- **Model-based Image Warping**

A key part of real-time video painting is to dynamically update the base texture buffer, which is achieved with a model-based image warping process. A base texture buffer is selected, and its pose (relative to the 3D model of the scene) is computed, which gives the correspondence between each pixel in the base buffer and a 3D model point. As shown in Figure 2, when video frames are processed, for a given pixel I_b in the base buffer we find the corresponding 3D point M . If the 3D point M is visible from the current camera viewpoint, it is then projected into the current camera image plane to find the corresponding pixel I_v , and we update the base buffer pixel I_b with the pixel I_v from the video frame. Figure 3 shows one video frame and the result of 3D model-based image warping into the base buffer.

- **Occlusion Detection**

Occlusion problems occur when the model is visible from the base buffer, but invisible (or partially visible) from the current

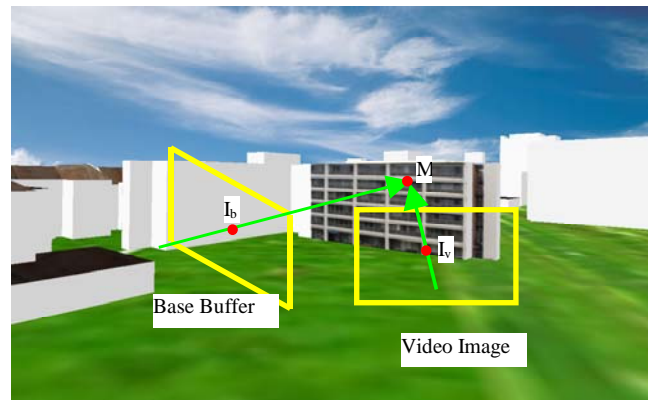


Figure 2. Model-based image warping.

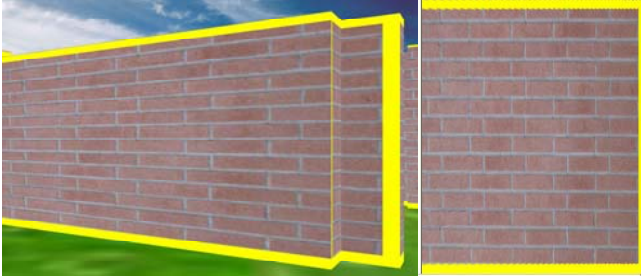


Figure 3. Result of model-based image warping. Left: original image. Right: image warped to the base buffer.

camera viewpoint. Direct model-based image warping will paint textures on these occluded model parts. As shown in Figure 4 (left), the model region inside the red ellipse is occluded by the model closer to the camera viewpoint; however, video painting without a depth test will paint textures on this part. The occlusion problem is resolved using a depth map test. We render a depth map of the 3D models for each camera position. As the image warp is processed, we find the corresponding 3D point for a pixel to be warped (Figure 2). Then we project the point back to the video image plane, and compare the Z-value with the current depth map. We keep the pixel information only if the Z-value is less than the corresponding depth. Figure 4 shows the result before and after the depth test.

- **Texture Segmentation**

Real-time video painting has the advantage of directly updating texture information to capture the most recent environment changes. However, if we don't select the content to be painted, it will paint everything in the video sequence onto the 3D environment, and possibly give an undesired result. To remove unwanted texture pixels of moving objects and objects without a pre-built model, we need to segment the video images. Video frames are first warped into the base buffer, and then a Gaussian method (or median method) is used to learn the 2D image background since all the images are registered with the base buffer after 3D warping. We then segment the image by subtracting the foreground textures from the background textures.

- **2D Image Registration**

The pose recovered from GPS (Global Positioning System) and inertial sensor or other tracking algorithms is usually not accurate

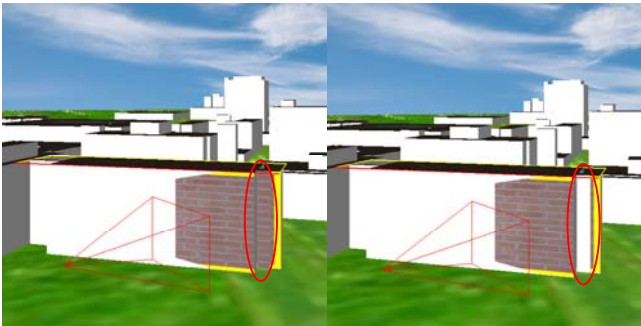


Figure 4. Occlusion processing. Left: video painting before a depth test. Right: video painting after a depth test. Notice the difference in the red ellipse region.



Figure 5. Refining texture alignment. Left: video painting using only 3D model-based warping, which results in significant misalignment due to an inaccurate camera pose. Right: video painting after refining the texture alignment.

enough for pixel precision image registration. Video painting using this pose information results in significant misalignment (Figure 5, left), which needs to be further refined to obtain better visual quality. A 2D image registration process is used to achieve seamless image alignment. We first warp the video frame based on the 3D model, and then register the warped image with the existing base buffer image using a 2D affine model (Equation 1).

$$\begin{bmatrix} I_{xb} \\ I_{yb} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} I_{xv} \\ I_{yv} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (1)$$

3. VIDEO PAINTING FOR A LARGE-SCALE ENVIRONMENT

The video painting method described in Section 2 is suitable for a few buildings [7]. However, to capture and update textures for a large-scale environment, such as a university campus, we need to overcome several challenges, including the automatic allocation of base buffers for each polygon, the dynamic detection of active base buffers, and real-time painting and rendering.

3.1 Base-buffer allocation

3.1.1 The Base-buffer allocation Problem

Given the 3D models of a large-scale environment, we need to allocate base buffers for texture storage. The goal of base-buffer allocation is to find an optimal allocation that covers all the geometry of the environment and satisfies three criteria: minimum number of base buffers, minimum number of clipped polygons, and optimal texture quality.

The number of base buffers corresponds to the number of texture units in scene rendering using graphics hardware, and is also proportional to the size of texture memory. Current graphics cards have a limited number of texture units and memory, so limiting the number of base buffers is necessary. Naïve ways to resolve the buffer-allocation problem, such as assigning one base buffer for each polygon surface, are infeasible because the number of base buffers will become prohibitive when the polygon count is large, e.g., a million (common in a large-scale environment).

Since the number of clipped polygons will increase the total number of polygons and affect the rendering speed, we also want to minimize this number. Polygon clipping is necessary when a base

buffer only covers part of a polygon. As shown in Figure 6, we need to clip the triangle in order to compute UV texture coordinates for the shown base buffer. Assigning base buffers that cover entire polygons will avoid the polygon-clipping problem.

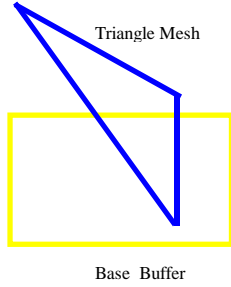


Figure 6. Polygon clipping.

The last issue with base-buffer allocation is the texture quality problem. Texture quality depends on the video resolution, the angle of the polygon relative to the camera view direction, and the angle of the polygon relative to the base buffer. A good allocation for a base buffer is one where textures will not be compressed or stretched during the texture mapping process. This means the base buffer should be oriented parallel to most of its associated polygons. Simple algorithms that combine nearby polygons into one base buffer often lead to low texture quality, since polygon orientation also has to be taken into account (Figure 7, left).

3.1.2 Previous Work

The problem of selecting optimal base buffers that cover the whole environment is analogous to the problem of selecting a minimal number of camera positions that cover the entire surface of a given 3D model. The latter problem has been shown to be NP-complete in computation geometry research [14]. When the environment model has a large number of polygons, the exact solution for optimal base-buffer allocation becomes prohibitive.

Matsushita [11] solves the camera-positioning problem using a heuristic that gives priority to viewpoints covering large model regions. This algorithm works well for an object, but has disadvantages in an environment with multiple buildings, as addressed in this research. First, many viewpoints will overlap, i.e., base buffers will overlap with each other, causing redundant texture coverage. Second, picking a viewpoint based only on the model coverage area could lead to base buffers that are not parallel to the model surfaces, thereby degrading the texture quality. Finally, polygon clipping is likely, since the viewpoint selection metric does not account for clipping (Figure 6).

3.1.3 Our Solution

We summarize the three criteria of base-buffer allocation using a cost function shown in Equation 2, where $f(bn)$ is a function of

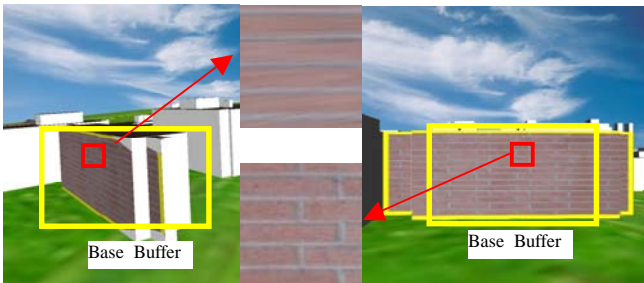


Figure 7. Texture quality. Left: low texture quality due to slanted angle of polygon relative to base buffer. Right: high texture quality when the base buffer is parallel to the polygon plane.



Figure 8. Base-buffer allocation. Each edge of the red rectangles denotes a base buffer.

the number of buffers, $g(pn)$ is a function of the number of clipped polygons, and h is a function of the texture quality. The design of the functions $f(bn)$ and $g(pn)$ is straightforward; however, the measurement of texture quality is more complex. Our measure is the ratio of the model surface area to the texture image area, as given in Equation 3. We aim to find a buffer allocation method that minimizes the cost of Equation 2.

$$F(\text{buffer}) = f(bn) + g(pn) + h(\text{texture_quality}) \quad (2)$$

$$h(\text{texture_quality}) = S(\text{polygon}) / S(\text{texture}) \quad (3)$$

Since the exact allocation solution is prohibitive, we attack this problem by leveraging the characteristics of the building models in the environment to find a near-optimal solution. First, since the roof and ground textures can best be acquired from an aerial image, we focus only on the textures of building facades. We note that most buildings are convex, and a cylindrical map is a simple solution that will cover most parts of most buildings. However, since buildings often have large planar surfaces, cylindrical texture mapping will stretch or compress the textures, degrading the texture quality. Noting that buildings usually have four major planar sides, we opt for using a box with four rectangular texture-maps for each building.

To allocate the four rectangular buffers, we first divide the scene model into separate buildings. Then we find the four major directions based on building surface normals and allocate a base buffer for each direction, as shown in Figure 8. To find the relationship between the model and base buffer, we project the vertices onto the base buffer with an orthogonal projection to compute the UV coordinate for each model vertex.

Four rectangular buffers work well for our campus model. The base buffers are automatically allocated, yielding a near-optimal solution according to the energy function in Equation 2. The campus model has 100 buildings and 40,000 polygons, covered by 400 base buffers. The texture quality is maximized for most polygons because the buffers are parallel to most of the polygon planes. Figure 7 (right) shows the base buffer relationship to building surfaces for producing high quality textures in our method. Since each base buffer covers one whole side of a building, polygon clipping is avoided.

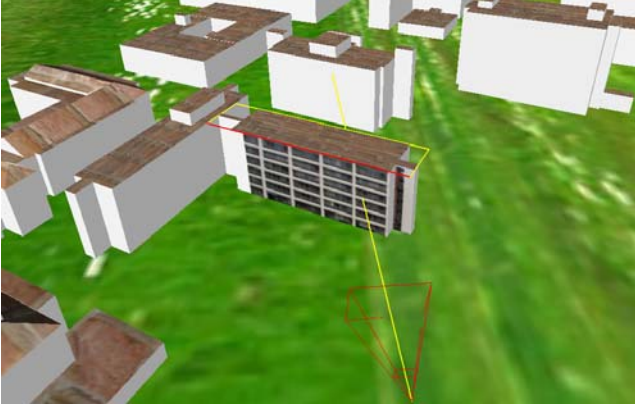


Figure 9. Detect active buffer. The yellow ray identifies the closest visible building. The red edge of the rectangle above the building denotes the current active buffer; the three yellow edges are the other three buffers of the active building.

Of course, the buffers assigned by this approach will not cover every polygon if the model is concave or there is surface occlusion. Fortunately, such cases are relatively rare, and more than 90 percent of the polygons are covered by the simple base buffers. To ensure that base buffers cover every polygon, we subdivide building surfaces into visible regions and clip any partially covered polygons.

3.2 Active Buffer Selection

Updating the textures for all base buffers at the same time is infeasible for a real-time system, so we update one base buffer (active base buffer) at a time.

A two-step strategy is employed to find the current active base buffer. First, we find all the buildings visible from the current camera viewpoint. To speed up this process, bounding boxes are computed for each building. Then these bounding boxes are intersected with the camera frustum to determine their visibility. Among all the visible buildings, we find the closest one to the current viewpoint (Figure 9). In a second step, we compute the angle between the buffer normal and camera optical axis for all four base buffers of the closest building, and set the buffer with the minimum angle as the current active buffer. If all the visible buffers in the closest building exceed an angle threshold, the next closest building is considered. A weighted average of the normalized building distance and buffer view angle is used to select the active buffer in order to ensure the texture quality (Equation 4).

$$w = \alpha * Dist + \beta * Angle \quad (4)$$

Where *Dist* is the minimum distance from the polygons of a building to the current viewpoint, *Angle* is the minimum angle between the buffers' normal and the view direction, and α and β are the weights. *Dist* and *Angle* are normalized respectively using the minimum distance and angle among all the visible buildings.

3.3 Implementation Details

3.3.1 Software Implementation

We first implemented the system in software by updating the base buffers pixel by pixel. The key process in video painting is 3D

model-based image warping, therefore we need to know the corresponding 3D model point for every pixel in each base buffer to do the image warping. In order to continuously update textures in real time, we pre-compute the depth map for each base buffer, and use it as a lookup table to find the 3D point for every pixel in the buffer. As a video frame is processed, we first find the active base buffer according to the aforementioned algorithm. For each pixel in the active buffer, we find the corresponding 3D point based on the depth map. If the 3D point passed the occlusion test (i.e., it is visible from the camera viewpoint), we project it into the current camera image plane to find the corresponding pixel. We then update the base buffer pixel with the pixel from the video frame. Textures are segmented if moving objects present, and the warped image is registered with the base buffer if the 3D pose is inaccurate. The system can reach up to 12 fps for video size of 256 x 256 (Table 1), given accurate pose.

The software implementation achieves real time for a medium-sized texture image, such as 512 x 512. However, we need to pre-compute and store a depth map for each base buffer, which costs additional memory. For our university campus with 100 buildings, each building has four buffers with a depth map of 512 x 512, so more than 400 MB are required to store the depth maps.

3.3.2 Hardware Implementation

Several issues in a practical video painting system require an efficient implementation that uses graphics hardware. First, we use hand-held cameras to capture video streams, which makes pose tracking a challenging task, usually not accurate enough to achieve seamless image alignment. Due to time-consuming 2D image registration, the video painting process is hard to achieve in real time using software implementation. On the other hand, high-resolution textures are required to achieve good visual effects. However, software implementation alone cannot satisfy the requirement. Finally, the storage of depth maps in software implementation is a heavy burden on the system for a large-scale environment. We need a more efficient implementation that uses graphics hardware to improve the performance and reduce the memory requirement.

Compared to 2D image warping, which can be easily implemented using GPU programming, image warping based on a 3D model is more complex to implement using graphics hardware. If the 3D model is just a plane, the relationship between the base buffer and camera image plane can be described using homography. However, image warping based on homography is impractical for our purpose because most building geometry is much more complex than a few planes. We can't afford to segment the video frames and warp them according to different homographies in a time-critical system.

Another candidate way to benefit from the graphics hardware is to render the scene with the current video frame as textures, then warp the rendered scene into the base buffer. However, classic texture mapping technique won't fulfill the requirement since the camera is continuously moving, which requires UV coordinate computing and polygon clipping for every frame.

To fully benefit from graphics hardware, we use the technique of projective texture mapping [10]. We first set the current active buffer as the rendering buffer, then render the current scene using projective texture mapping; we solve the visibility problem using

a hardware register combination based on a depth map. In order to update the buffer with new textures and keep the old textures, we use an alpha channel to mask out all the pixels outside the camera frustum and those pixels without textures. The process is summarized using the following pseudo code:

1. *Dynamically detect the active base buffer.*
2. *Render a depth map from the camera viewpoint. (The depth map here is used to solve the visibility problem, not to find the correspondence between model points and pixels in the base buffer.)*
3. *Load the image of the current base buffer (with textures already painted) into the rendering buffer.*
4. *Render the scene using projective texture mapping with the current video frame as the textures (no pre-computed depth map from the base buffer is required), and solve the visibility problem based on a depth map from the camera's viewpoint. Set the alpha value of the pixels without textures or invisible to zero.*
5. *Enable alpha test; only render pixels with alpha value greater than zero into base buffer.*
6. *Read the rendering buffer, and set it as the textures for the models corresponding to the current base buffer.*

We can gain many benefits through the implementation of using graphics hardware. The frame rate increases dramatically, and we can achieve real time for high-resolution textures of 1024 x 1024. Using hardware implementation, depth map storage for base buffers is not necessary, which saves a great deal of memory. Table 1 compares the frame rate at different sensor resolutions and memory requirements (for a fixed buffer resolution of 512 x 512 and 100 buildings) for both software and hardware implementation on a Dell machine with 3.4 G CPU and NVIDIA quadro 980 graphics card.

4. RESULTS

• Simulation Data

We first tested the presented system using simulation data. LiDAR data was used to model a university campus with about 100 buildings [22]. The campus model was texture-mapped with brick textures to create a simulated environment (Figure 10, left). A simulated camera (shown with a red frustum in Figure 10) moved freely to capture the video, which was then painted onto

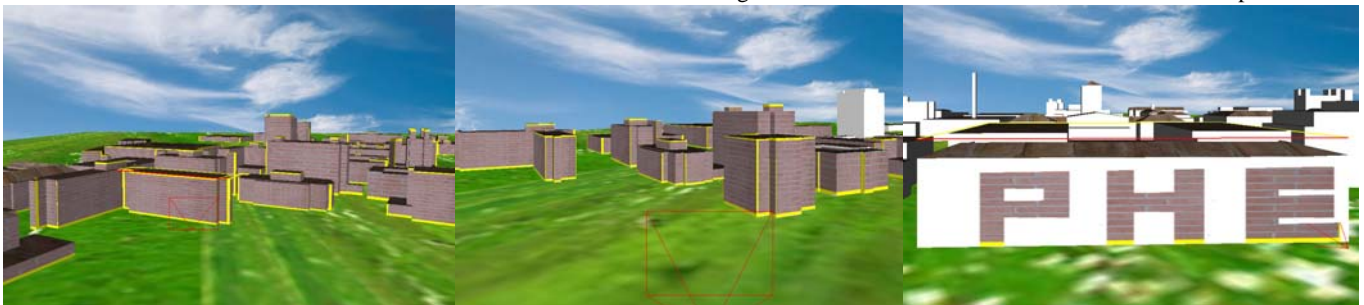


Figure 10. Simulated data. Left: simulation environment. Middle: tens of buildings painted in several minutes. Right: user-painted letters on building surfaces.

Table 1. Frame rate and memory requirements of hardware and software implementation

Method	Frame rate / sensor resolution			Memory
	256	512	1024	
Software	12 fps	4 fps	1 fps	700 M
Hardware	30 fps	20 fps	10 fps	300 M

the 3D model using the presented approach. In the simulation data, since the camera pose was perfect, no further 2D image registration was needed. The whole system was real-time using a 3.4GHZ DELL machine (Table 1). We found that most of the time was spent rendering the whole scene with texture mapping rather than painting. When we only rendered the current building being painted, the frame rate reached up to 60fps, so we could paint multiple video streams at the same time in real time. Figure 10 (middle) shows the result of tens of buildings painted in several minutes.

Unlike traditional 2D image registration, which requires image overlap, 3D model-based image registration requires no constraints on camera motion. With a specific camera motion path, we designed special shapes using the given texture. Figure 10 (right) shows the PHE letter painted onto the surface of buildings using this technique. Other irregular shapes were also painted, which is hard to achieve using traditional texture mapping methods. This feature also makes the video painting technique a potential method for special effects and artistic design.

• Real Data

We used real video streams captured from hand-held cameras to further test the algorithms. While painting textures that cover a whole simulated environment is easy (we accomplished it in less than a half-hour), capturing video data that covers a whole real environment (with the same models) is much harder, even with hand-held cameras. More research is needed to explore techniques for camera orienting and path design to efficiently capture data, which is out of the scope of this research.

The strengths of our system are automatic texture mapping and real-time texture updating; to show this, we captured three video streams on a university campus with different motions. The motion of the first video stream is mainly rotation, the second translation, and the third both translation and rotation. A portable tracking and video system [12] was used to collect camera tracking data and video streams. The video streams were painted onto



Figure 11. Painting results of three real video streams. Upper row: selected frames from the three video streams. Lower row: painting result.

the 3D models using the proposed system. Figure 11 shows the result (further details are provided in the video clip accompanying this paper). The whole system is fully automatic and real-time (29 fps, thanks to hardware implementation).

5. CONCLUSION

Creating a large-scale photorealistic environment is important for many virtual reality applications. However, the acquisition of static textures for such a large-scale environment is a challenging task using traditional techniques, which often demand tedious and time-consuming manual interactions. Texture updating using the classic method is even more challenging for such a large environment. This paper presents a practical and real-time system that is capable of acquiring and updating textures for a large-scale environment through video painting. Using the proposed techniques, we can not only create textures for a university campus-sized environment, but also update the textures in real time.

While we have shown the presented techniques' success in a university campus-sized environment, there are still some research problems that need to be addressed:

- *3D model-based pose refinement.* Although we used 2D registration to refine the pose, the result depends on the pose of the first frame warped to the base buffer. To further improve the visual effect, we need to improve the 3D pose of the first frame warped to the base buffer.
- *View-dependent rendering.* As we have mentioned, the rendering of the whole scene with texture mapping takes more time than the actual painting process. So some rendering techniques, such as view-dependent rendering, are necessary to guarantee the real-time rendering for models with high-resolution textures.

6. ACKNOWLEDGEMENT

This work was supported by the National Geospatial Intelligence Agency (NGA) under a NGA University Research Initiative (NURI) program, and in part by a Multidisciplinary University Research Initiative (MRUI). We thank the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, for its support and facilities.

7. REFERENCES

- [1] Ashikhmin, M. Synthesizing natural textures. *ACM Symposium on Interactive 3D Graphics*, 2001, 217–226.
- [2] Berman, D.F. et al. Multiresolution painting and compositing. *Proceedings of SIGGRAPH 94*, 1994.
- [3] Bernardini, F., Martin, I. M., and Rushmeier, H. High-quality texture reconstruction from multiple scans. *IEEE Visualization and Computer Graphics*, Volume 7, Issue 4, 2001, 318–332.
- [4] Efros, A., and Leung, T. Texture synthesis by non-parametric sampling. *ICCV*, 1999, 1033–1038.
- [5] Efros, A., and Freeman, W. T. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001*, 2001, 341–346.
- [6] Heeger, D.J., and Bergen, J.R. Pyramid-based texture analysis/synthesis. *Proceedings of SIGGRAPH 95*, 1995, 229–238.
- [7] Hu, J., You, S., and Neumann, U. Texture painting from video. *Journal of WSCG, ISSN 1213–6972, Volume 13, Number 1-3*, 2005, 119–125.

- [8] Igarashi, T., and Cosgrove, D. Adaptive unwrapping for interactive texture painting. *ACM Symposium on Interactive 3D Graphics*, 2001.
- [9] Laycock, R.G., and Day, A.M. Automatic techniques for texture mapping in virtual urban environments. *Computer Graphics International*, 2004, 586–589.
- [10] Mark, S. et al. Fast Shadows and lighting effects using texture mapping. *Siggraph'78*, 1978, 270–274.
- [11] Matsushita K., and Kaneko, T. Efficient and handy texture mapping on 3D surfaces. *In Proc. of Eurographics*, 1999, 349–358.
- [12] Neumann, U., You, S., Hu, J., Jiang, B., and Lee, J. W. Augmented virtual environments (AVE): dynamic fusion of imagery and 3D models, *IEEE Virtual Reality*, Los Angeles, California, 2003, 61–67.
- [13] Ofek, E. et al. Multiresolution textures from image sequences. *IEEE Computer Graphics and Applications*, Volume 17, Issue 2, 1997, 18–29.
- [14] O'Rourke, J. *Art Gallery Theorems and Algorithms*, Oxford University Press, New York, 1987.
- [15] Perlin, K. Live paint: painting with procedural multiscale textures. *Proceedings of SIGGRAPH*, 1995, 153–160.
- [16] Portilla, J., and Simoncelli, E.P. A parametric texture model-based on joint statistics of complex wavelet coefficients. *IJCV* 40, 1(Oct.), 2000, 49–70.
- [17] Rocchini, C., Cignoni, P. and Montani, C. Multiple textures stitching and blending on 3D objects. *In Eurographics Rendering Workshop*, 1999.
- [18] Schodl, A., Szeliski, R., Salesin, D. H., and Essa, I. Video textures. *Proceedings of SIGGRAPH*, 2000, 489–498.
- [19] Soatto, S., Doretto, G., and Wu, Y. Dynamic textures. *In Proceeding of IEEE International Conference on Computer Vision*, II, 2001, 439–446.
- [20] Wang, X., Totaro S., Taillardier, F., Hanson, A., and Teller, S. Recovering façade texture and microstructure from real world images. *In Proceedings 2nd International Workshop on Texture Analysis and Synthesis at ECCV*, 2002, 145–149.
- [21] Wei, L. Y., and Levoy, M. Fast texture synthesis using tree structured vector quantization. *Proceedings of SIGGRAPH*, 2000, 479–488.
- [22] You, S., Hu, J., Neumann, U., and Fox, P. Urban Site Modeling From LiDAR, *Second International Workshop on Computer Graphics and Geometric Modeling*, 2003.