International Journal of Information Technology
Vol. 8, No. 2 September

Texture Mapping of 3D Human Face for Virtual Reality Environments

Alvin W. K. Soh, Zhang Yu, Edmond C. Prakash, Tony K. Y. Chan and Eric Sung

School of Computer Engineering, Nanyang Technological University
Singapore - 639 798 Email: Asprakash@ntu.edu.sg

Abstract

Texture mapping of 3D digitized data is a challenge. This arises mainly due to two factors: i) ***digitization of geometry*** *and ii)* ***digitization of textures****. First, due to* ***digitization of geometry*** *we have a basket of polygons that represent the digitized object. Second, due to* ***digitization of textures****, we get another basket of textures that represent the digitized texture of the whole object. Unfortunately, there is little or no correspondence between the digitized geometry and digitized texture. The challenge is to automatically identify the correspondence between the merged geometry and merged texture and then to perform an intelligent texture mapping. In this work, we have shown two procedures that perform 3D texture mapping using a combination of image warping and direct texture mapping.*

Keywords: *Texture Mapping, Virtual Reality, Virtual Environments, 3D Face Modeling*

1. Introduction

Creating compelling and detailed graphic content to take advantage of powerful hardware and software platform have been the focus of research in graphics community in recent years. Traditional techniques for generating content have been extremely expensive and laborious. Most of the software tools created for generation of 3D models and textures expect humans to enter the details of the 3D models manually using the graphical user interface of the packages. In recent years, the numbers of range scanners and surface reconstruction algorithms have been growing rapidly. Current efforts have concentrated on automatic capturing of detailed geometric objects by direct observation of the real life objects. These techniques "scan" 3D objects and their attributes to obtain sufficient information to render them in a digital environment.

Digitization of 3D models is now feasible via active 3-D laser scanners, digital interface/progressive scan CCD cameras, high-resolution digital still video, and integrated different types of data. 3D scanning is similar in principle to a number of other important technologies (like photocopying and video) that quickly, accurately, and cost effectively record useful aspects of physical reality. These technologies have had an enormous impact, primarily because electronic representations can be used in ways the original physical objects cannot. For example, they can be stored in, searched for, and retrieved from databases, transmitted electronically over long distances, viewed on CRTs, used in computer simulations, manipulated and edited in software, and used as templates for making electronic or physical copies.

3D scanning is roughly in the same state as photocopying before the invention of xerography, or home movies before the invention of compact video cameras. By analogy, it is reasonable to expect that the development of the technology will open up many new applications, in areas such as reverse engineering, industrial design, repair, reproduction, improvement of machinery; medical diagnostics, analysis and simulation, 3D photography; and building rich virtual environments. 3-D Digitization is a method for acquiring and processing of such three-dimensional data obtained from a 3-D scanner. This growing field has many applications in robotics, industrial design and inspection, human engineering, medical and in imaging sciences.

2. Human Face Modeling

2.1 Model Acquisition And Rendering Techniques

The major digitization schemes currently under use includes:

1. 3D Digitizers: To obtain 3D digital models of existing real world objects or to obtain such data from clay models, 3D stylus-based digitizers are commonly used. Once this data is obtained, various software techniques are used to either retain the digitized data as a collection of bilinear patches or a higher order surface is fitted on to this data.
2. 3D Laser Scanners: Laser range scanning devices based on light interferometry provides a much more automatic tool for obtaining a digital model of an existing 3D object. These scanners are rotated around a given object at a specific step size and a large number of measurements are taken. These measurements often consist of geometric position as well as the texture information at the scanned point. The scanning process gives a complete description needed to render the object. Depending on the application, the data is converted to either a triangle mesh or a quadrilateral providing a piecewise bilinear model of the surface.
3. 3D Range Scanners: Range scanning devices are basically digital cameras that capture, for each pixel, the distance of the closest object. A single snapshot, however, gives the 3D information only from one viewpoint. To capture the 3D geometry from all directions around the 3D object of interest, multiple pictures need to be taken with substantial overlap across the images. Such range images are combined to form a new complete mesh representing the scanned object.
4. Image based acquisition of 3D models and textures: Recovery of 3D geometry from a set of images snapped from known or unknown viewpoints has been used to acquire 3D geometry data from images. Viewing these models from arbitrary viewpoints is now possible since the 3D model is available. However, the texture mapping must be view dependent. This is because the textures captured from images of the 3D objects will not have enough information about the occluded geometry. Hence as the viewpoint changes with respect to the reconstructed geometry, the model’s attributes like texture need to be dynamically changed.
5. Image-based rendering: In image-based rendering techniques, no attempt is made to reconstruct the 3D geometry of the actual 3D world objects, but the captured images are warped, transformed, clipped and pasted in appropriate places to generate an imagery that gives an illusion of 3D scene navigation as the viewing parameters of the viewer changes. The aim is to gather enough data about the model to be able to create a reasonably realistic reproduction of the 3D object from an arbitrary viewpoint and possibly different possible illumination conditions.

2.2 File Formats And Conversion

In the absence of an industry-standard 3D file interchange format, several de facto 3D file standards have appeared for the Macintosh, PC, Amiga, and UNIX platforms. Probably the most popular formats include DXF, 3DS, VRML, etc. Developers of software that needs to be able to import or export 3D information in a way that is useful for many potential users still must pick from dozens of existing formats

The 3D models are acquired using various processes and stored in forms that are different from those required for the end use. The data needs to be converted to a suitable file format and the model also needs to be transformed so that it is in a usable state. Listed below are issues related to the file formats and transformation of models for the end application:

- Information captured in the source file format may not be sufficient to construct the details necessary to build the model in the other file format. In such cases, it may be necessary to infer this information.
- The conversion between file formats may be a lossy process where the target file format does not need the excess data in the source file format.
- The data types used in the source file format and the destination file format may not be the same. Hence some approximation of model may have to be accommodated.
- The source data may have a very high resolution that may not be suitable for the end application and needs to be reduced. While reducing the data, the information being communicated (from the viewpoint of the application) must be minimally affected. This can be achieved by adaptive sampling rather than by trivial uniform sampling.
- The source data sometimes has errors incurred during recording or modeling of objects, either due to the instrumentation errors or human errors. These errors could be missing polygons, gaps in the geometric description, incorrect textures mapped onto the geometry. These artifacts must be corrected before using the models for the application. Some of the errors can be healed automatically, where some need to be manually fixed by loading the model in some editor.
- To achieve faster transmission and progressive loading of the models, the data must be transformed into a suitable format such that the information is differently represented. Researchers have proposed many model compression and progressive transmission schemes.

2.3 Texture Mapping

Texture mapping onto 3D models has been the topic of recent research [1,2,4] even though early work appeared in the late seventies [5]. Texture mapping is a powerful technique for adding realism to a computer-generated scene. In its basic form, texture mapping lays an image (the texture) onto an object in a scene. More general forms of texture mapping generalize the image to other information; an "image" of altitudes, for instance, can be used to control shading across a surface to achieve such effects as bump mapping. Because texture mapping is so useful, it is being provided as a standard rendering technique both in graphics software interfaces and in computer graphics hardware. Texture mapping can therefore be used in a scene with only a modest increase in the complexity of the program that generates that scene, sometimes with little effect on scene generation time. The wide availability and high-performance of texture mapping makes it a desirable rendering technique for achieving a number of effects that are normally obtained with special purpose drawing hardware.

When mapping an image onto an object, the color of the object at each pixel is modified by a corresponding color from the image. In general, obtaining this color from the image conceptually requires several steps. The image is normally stored as a sampled array, so a continuous image must first be reconstructed from the samples. Next, the image must be warped to match any distortion (caused, perhaps, by perspective) in the projected object being displayed. Then this warped image is filtered to remove high-frequency components that would lead to aliasing in the final step, resampling to obtain the desired color to apply to the pixel being textured.

In practice, the required filtering is approximated by one of the several methods. One of the most popular approach is mip-mapping. Other filtering techniques may also be used. There are a number of generalizations to this basic texture-mapping scheme. The image to be mapped need not be two-dimensional; the sampling and filtering techniques may be applied for both one- and three-dimensional images. In the case of a three-dimensional image, a two-dimensional slice must be selected to be mapped onto an object's boundary, since the result of rendering must be two-dimensional. The image may not be stored as an array but may be procedurally generated. Finally, the image may not represent color at all, but may instead describe transparency or other surface properties to be used in lighting or shading calculations.

1. Basic Texture Mapping: In basic texture mapping, an image is applied to a polygon (or some other surface facet) by assigning texture coordinates to the polygon's vertices. These coordinates index a texture image, and are interpolated across the polygon to determine, at each of the polygon's pixels, a texture image value. The result is that some portion of the texture image is mapped onto the polygon when the polygon is viewed on the screen.

2. Projective Textures: A generalization of this technique projects a texture onto surfaces as if the texture is a projected slide or movie. In this case the texture coordinates at a vertex are computed as the result of the projection rather than being assigned fixed values. This technique may be used to simulate spotlights as well as the reprojection of a photograph of an object back onto that object's geometry.

3. Image Warping: Image warping may be implemented with texture mapping by defining a correspondence between a uniform polygonal mesh (representing the original image) and a warped mesh (representing the warped image). The warp may be affine (to generate rotations, translations, shear, and zoom) or higher-order. The points of the warped mesh are assigned the corresponding texture coordinates of the uniform mesh, and the mesh is texture mapped with the original image. This technique allows for easily controlled interactive image warping. The technique can also be used for panning across a large texture image by using a mesh that indexes only a portion of the entire image.

4. Transparency Mapping: Texture mapping may be used to lay transparent or semi-transparent objects over a scene representing transparency values in the texture image as well as color values. This technique is useful for simulating clouds and trees for example, by drawing appropriately textured polygons over a background. The effect is that the background shows through around the edges of the clouds or branches of the trees. Texture map filtering applied to the transparency and color values automatically leads to soft boundaries between the clouds or trees and the background.

5. Environment Mapping: Environment mapping may be achieved through texture mapping in one of two ways. The first way requires six texture images, each corresponding to a face of a cube that represent the surrounding environment. At each vertex of a polygon to be environmentally mapped, a reflection vector from the eye to the surface is computed. This reflection vector indexes one of the six texture images. As long as all the vertices of the polygon generate reflections onto the same image, the image is mapped onto the polygon using projective texturing. If a polygon has reflections into more than one face of the cube, then the polygon is subdivided into pieces, each of which generates reflections onto only one face. Because a reflection vector is not computed at each pixel, this method is not exact, but the results are quite convincing when the polygons are small.

2.4 Equipment (Vivid 700)

The Vivid 700 3D-laser scanner from Minolta has been used in our work. The main features includes:

1. Easy Stand-alone operation: The VIVID captures data independent of any host computer with ease using a point and shoot camera. The optional memory card allows self-contained scanning and data storage. When used with the optional rotating stage, scans are aligned automatically for especially fast model making.
2. High-speed Scanning: It has a 0.6 second scan time.
3. Portable: At under 20 lbs. The VIVID 700 is easily brought to the subject and repositioned for each scan.
4. Powerful Editing Tool Suite: Standard equipment: Using a combination of software and hardware the scanning system 'knows' where in 3D space the reflected laser light is. Thus by sweeping a laser over an entire object the 3D shape can be created in digital form.

2.5 Vivid Texture Mapping

Figure 1. shows snapshots of three different faces captured and texture mapped using VIVID. Unfortunately, these textures and models cannot be used directly in any of our applications. Even though vivid could do texture mapping for one view at a time, when it is exported to a new program, there is no correspondence between the exported texture with the whole 3D object. Another problem is that we need the texture mapping for whole 3D object and not for just one view when we render it in different software.

To solve the above problem, we have developed several texture mapping techniques. Two successful techniques are described in detail in this paper. They are:

1. Single-Texture Mapping for Human Face Model
2. Multiple-Texture Mapping for Human Face Model

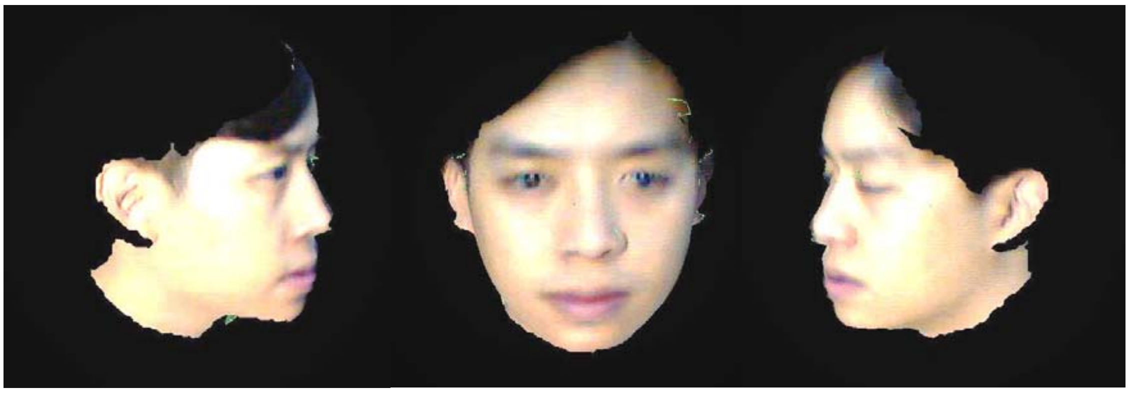


Fig. 1. Texture mapping of a single view using Vivid

3. Single-Texture Human Face Model

3.1 Texture Mapping Using Open Inventor

In the first approach we use the Open Inventor library, is a collection of graphics functions developed by Silicon Graphics to support easier use of the high-quality, high-performance 3D graphics (OpenGL) on their workstations that otherwise would require substantial programming experience. It is an object-oriented 3D toolkit offering a comprehensive solution to interactive graphics programming problems. It presents a programming model based on a 3D scene database that dramatically simplifies graphics programming. It includes a rich set of objects such as cubes, polygons, text, materials, cameras, lights, trackballs, handle boxes, 3D viewers, and editors that speed up your programming time and extend your 3D programming capabilities. ***Our contribution here is to demonstrate that acceptable quality can be achieved for simple real-time applications using existing library functions, in this case the Open Inventor Library.***

3.2 Pseudo Code of Texture Mapping Using Open Inventor

The pseudo code for texture mapping is shown below.

```
main(int , char **argv)
{
  char source[20];                        // IV FILENAME
  char tex[20];                           //RGB FILENAME

  Read open inventor file;
  Read the texture rgb file name;

  SoSeparator *readFile(const char*);     //INITIALISE
  Widget myWindow = SoXt::init(argv[0]);
  SoSeparator *root = new SoSeparator;    //DEFINE ROOT
  root→ref();
```

```
    SoTexture2 *face = new SoTexture2;          //DEFINE TEXTURE CLASS
    Root→addChild(face);                        //ADD TEXTURE TO ROOT SUB-BRANCH
    Face→filename = tex;                        //READ IN STORED RGB FILE NAME
    Root→addChild(readFile(source));            //READ IN IV FILE NAME

    //SETUP VIEWER
    SoXtExaminerViewer *myViewer = new
    SoXtExaminerViewer(myWindow);
    MyViewer→setSceneGraph(root);
    //myViewer→setTitle("3dview → ");
    myViewer→setTitle(source);
    myViewer→show();

    SoXt::show(myWindow);
    SoXt::mainLoop();
}

//READ IN FILE INTO DATABASE
    SoSeparator *
    readFile(const char *filename)
    {
       // Open the input file
       SoInput mySceneInput;
       mySceneInput.openFile(filename);

       // Read the whole file into the database
       SoSeparator *myGraph = SoDB:: readAll(&mySceneInput);
       mySceneInput.closeFile();
       return myGraph;
    }
```

After the human head has been scanned into vivid, the data of the elements are exported into Open Inventor *iv format. The image files are exported into *tiff files. Using XV file converter, the *tiff files are converted into *rgb files. A C program is written to render the *iv files for the human head and do texture mapping using the *rgb image files. As the texture mapping is done using only one *rgb image file to wrap around the whole head, an image editor, is used to morph a few images together. Figure 2(a) and 2(b) shows the results from three views of our texture mapping system.

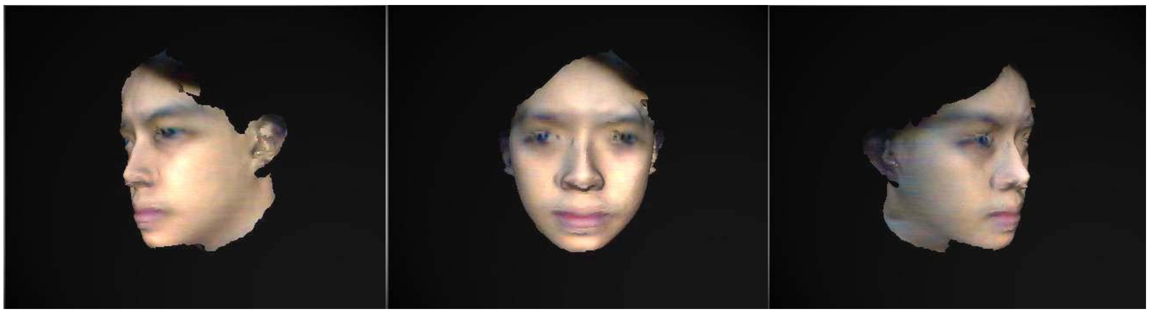


Fig. 2a. Sample Texture Mapping using Open Inventor (face1 showing three views)

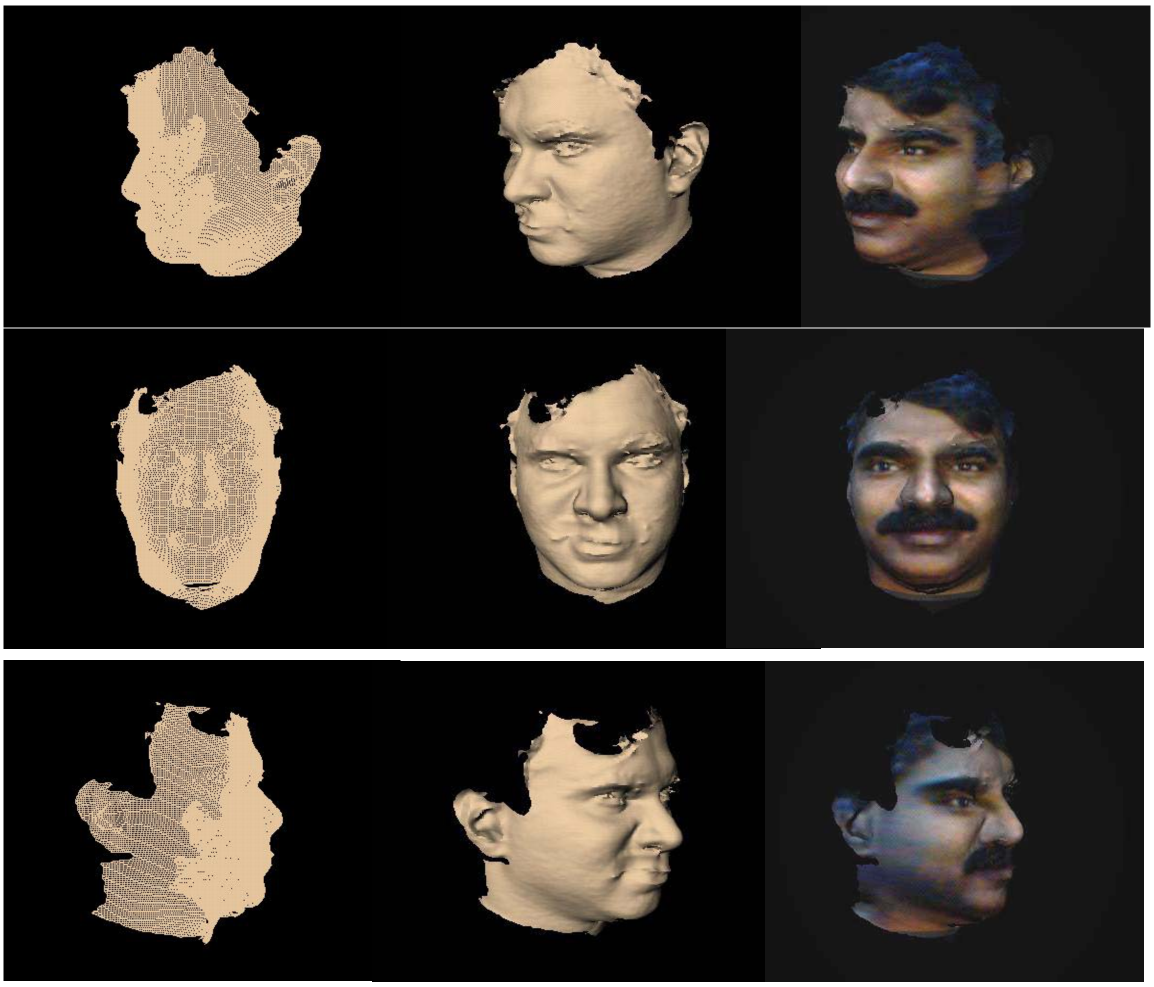


Fig. 2b. Texture Mapping using Open Inventor (face2 showing wireframe, shaded & textured in three views)

4. Multiple-Texture Human Face Model

4.1 Automatic View-based Texture Definition

To increase the realism, we utilize texture mapping. We have a discrete set of reflectance color images acquired by the 3D digitizer from different viewpoints. In general, each image views only a piece of the face. The images of Fig. 3 show two different reflectance images mapped onto the model and rendered from a novel viewpoint. Some triangles are colored in just one of the renderings, while some are colored in both. Thus, it is necessary to use multiple images in order to render the entire model from a novel point of view. Unlike the method used in the previous work [6, 7], which generates a composite texture map by combing several different views of the person offline, we blend the acquired multiple reflectance images by adjusting the blending weights dynamically, according to the current view.

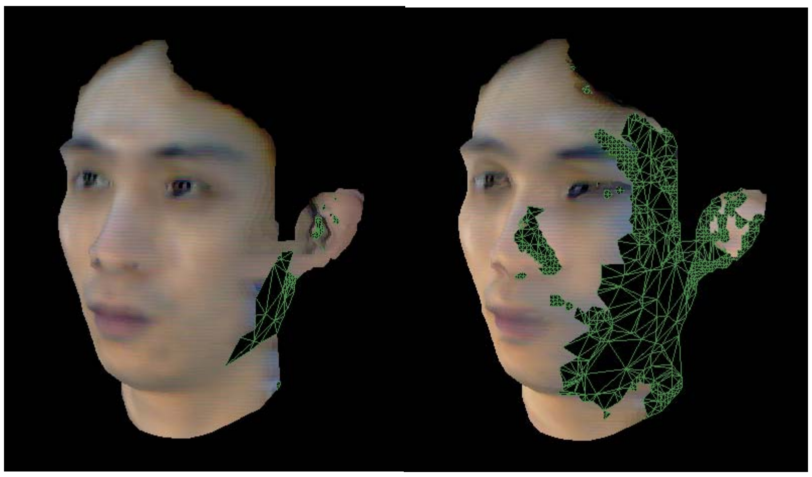
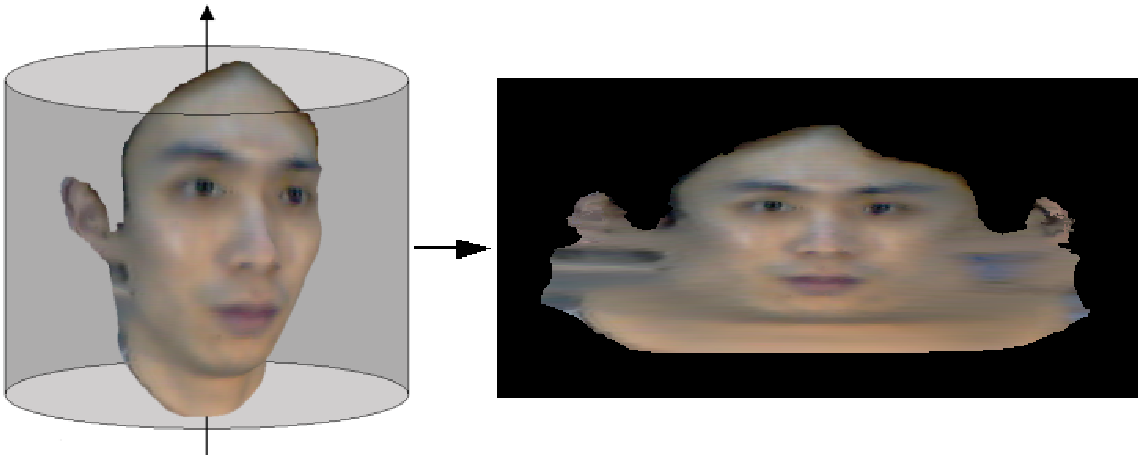
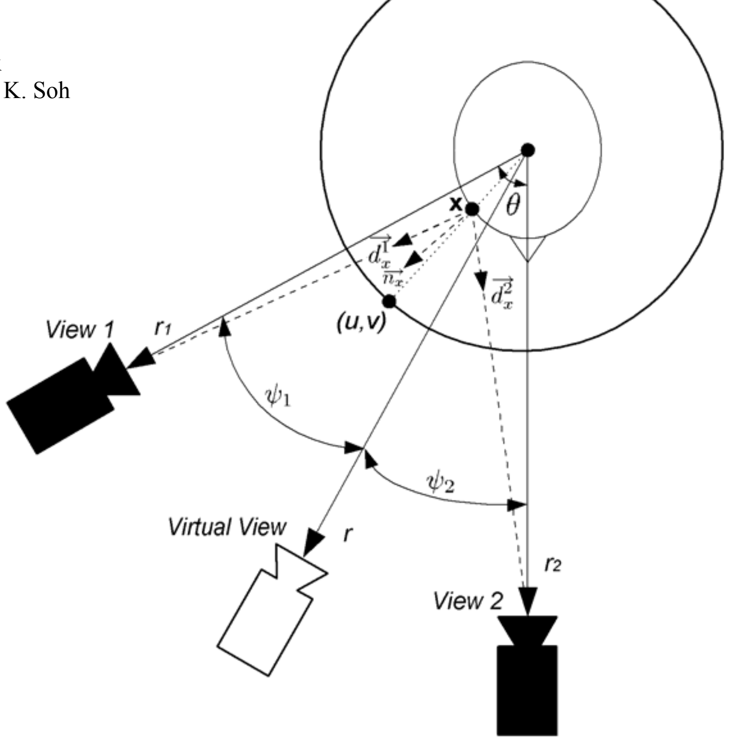


Figure 3: A novel view of the reconstructed face mesh with two different reflectance images mapped from their respective view directions.



(a)



(b)

Figure 4: (a) Cylindrical projection of the rendered facial mesh; (b) View-based texture blending by cylindrical projection.

4.2 Multiple-Texture Blending

To synthesis an image of the face from a fixed viewing direction, we first render the reconstructed face model several times, each time using a different input reflectance images $S_i, i = 1,\text{K} \ ,n$ as a texture map. By using a cylindrical projection, a 512×256 cylindrical texture map P_i corresponding to the face mesh individually rendered by each input image S_i is constructed on a virtual cylinder enclosing the face model (see Fig. 4 (a)). Given a viewing direction, we then select the subset of the cylindrical texture maps for texture blending. The images are blended into a single comprehensive texture map by the following weighting scheme:

$$I(u,v) = \frac{\sum_{i=1}^{n_s} W_i(u,v)P_i(u,v)}{\sum_{i=1}^{n_2} W_i(u,v)} \qquad (1)$$

Here, *I* is the final blended texture map, (*u,v*) are the image coordinates of each pixel, n_s is the number of cylindrical texture maps involved in blending, and P_i are the cylindrical texture maps. The weight function $W_i(u,v)$ is designed to specify the contribution of the *i*-th cylindrical texture map to the final blended texture. We set the blending function as the product of three weight terms:

$$W_i(u,v) = W_{i,\psi} W_{i,\varpi}(u,v) W_{i,o}(u,v) \qquad (2)$$

The definition of the weight terms is shown in Fig. 4 (b). In this figure, the ellipse represents the top view of the 3D face model. The wedge attached on the bottom of the ellipse indicates the nose - that is, the lower part of the ellipse is the front of a face. The large circle represents the cylindrical mapping plane onto which the 3D face model projected. Given a virtual view with view direction r, the blending weights of two actual views whose view directions r_1 and r_2 are the nearest to r will be computed. In the texture blending, we wish to use just a single cylindrical texture map if its view direction matches the current view direction precisely, and to blend smoothly between the closest two cylindrical texture maps otherwise. For this effect, we define the first term

$$W_{i,\psi} = \begin{cases} \cos(\frac{\psi}{\theta} \cdot \frac{\pi}{2}) & \text{if r is between } r_1 \text{ and } r_2 \\ 0 & \text{otherwise} \end{cases} \qquad (3)$$

where ψ_i i is the angle between the viewing directions of the virtual and actual views, θ is the angle between two actual views. This weight term measures the proximity of the stored actual views to the current viewpoint, and therefore changes dynamically as the virtual viewpoint moves. Specifically, $W_{i,\psi}$ guarantees the use of single cylindrical texture map for blending when the view directions of virtual and actual views are the same.

Since color and surface geometry is sampled much more densely at surface locations that are perpendicular to the sensor than at tilted surfaces and the range data is usually less reliable at tilted surfaces, each pixel (*u,v*) in the *i*-th cylindrical texture map P_i should be nonuniformly weighted depending on the position of its corresponding point on the face mesh. The second weight term, $W_{i,\varpi}$, is a static measure of the surface sampling density:

$$W_{i,\varpi}(u,v) = \overrightarrow{n_x} \cdot \overrightarrow{d_x^i} \qquad (4)$$

where $\overrightarrow{n_x}$ is the surface normal at facial point **x** whose cylindrical projection is (*u,v*), and $\overrightarrow{d_x^i}$ is a unit vector pointing from **x** to the *i*-th actual viewpoint (see Fig. 4 (b)). It is essential to compute the 3D point **x** on the surface of the face mesh to determine the value of $W_{i,\varpi}$ for each pixel. This computation is performed by casting a ray from pixel (*u,v*) on the cylinder towards the cylinder's axis. The first intersection of this ray and the face mesh is the point **x**.

The effect of self-occlusion should be minimized, that is, $W_i(u,v)$ should be zero unless **x** is front facing with respect to the *i*-th cylindrical texture map and visible in it. The third weight term $W_{i,o}$ represents a binary visibility map for each cylindrical texture map P_i to handling self-occlusion. These maps are defined in the same cylindrical coordinates as P_i. To verify if a facial point **x** corresponding to a pixel $P_i(u,v)$ is visible in an actual view we must check that: the normal vector in **x** is directed towards the viewpoint, and there are no other intersections of the facial mesh with the line that connect **x** and the viewpoint. For each actual view we first compute the angle between the normal of each point and the actual view direction r_i, which is called *backface culling angl*e. Then we compute the position of each point on the face model by applying viewing transformation. By consulting depth information from the Z-buffer, we are able to test possible mesh intersections on the line of sight. To classify the visibility of each facial point, if its backface culling angle is lower than π/2 and it has the smallest z-value, the corresponding point of $P_i(u,v)$ is visible, and the weight $W_{i,o}(u,v)$ is set to 1. Otherwise its value is set to 0.

For a real-time implementation, since two of the weight terms, $W_{i,\varpi}$ and $W_{i,o}$, do not depend on the view direction of the virtual view, we apply both of them offline and code them into the alpha channel of the cylindrical texture map. In the interactive viewing of the face model, for each virtual view, we calculate the dot product of the view directions for the stored actual views and the view direction of the current virtual view. The two views with the highest dot product values (the weight $W_{i,\psi}$) are then chosen for blending. When we render the facial mesh with the described colors, the hardware calculates the correct weights for us. The alpha value in each pixel is $W_{i,\varpi}$ $W_{i,o}$. It is also possible to apply $W_{i,\psi}$; using graphics hardware. After we render the views, we have to read in the information from frame buffer. OpenGL allows scaling each pixel while reading the frame buffer into memory. We can scale the alpha channel by $W_{i,\psi}$. The resulting alpha value becomes the total weight $W_i = W_{i,\psi} W_{i,\varpi} W_{i,o}$.

4.3 Multiple-Texture Results

Fig. 5 shows the view-based texture extraction and mapping for rendering the facial model from the view direction of a virtual view. In the top row color images of the subject taken from two actual views are shown.

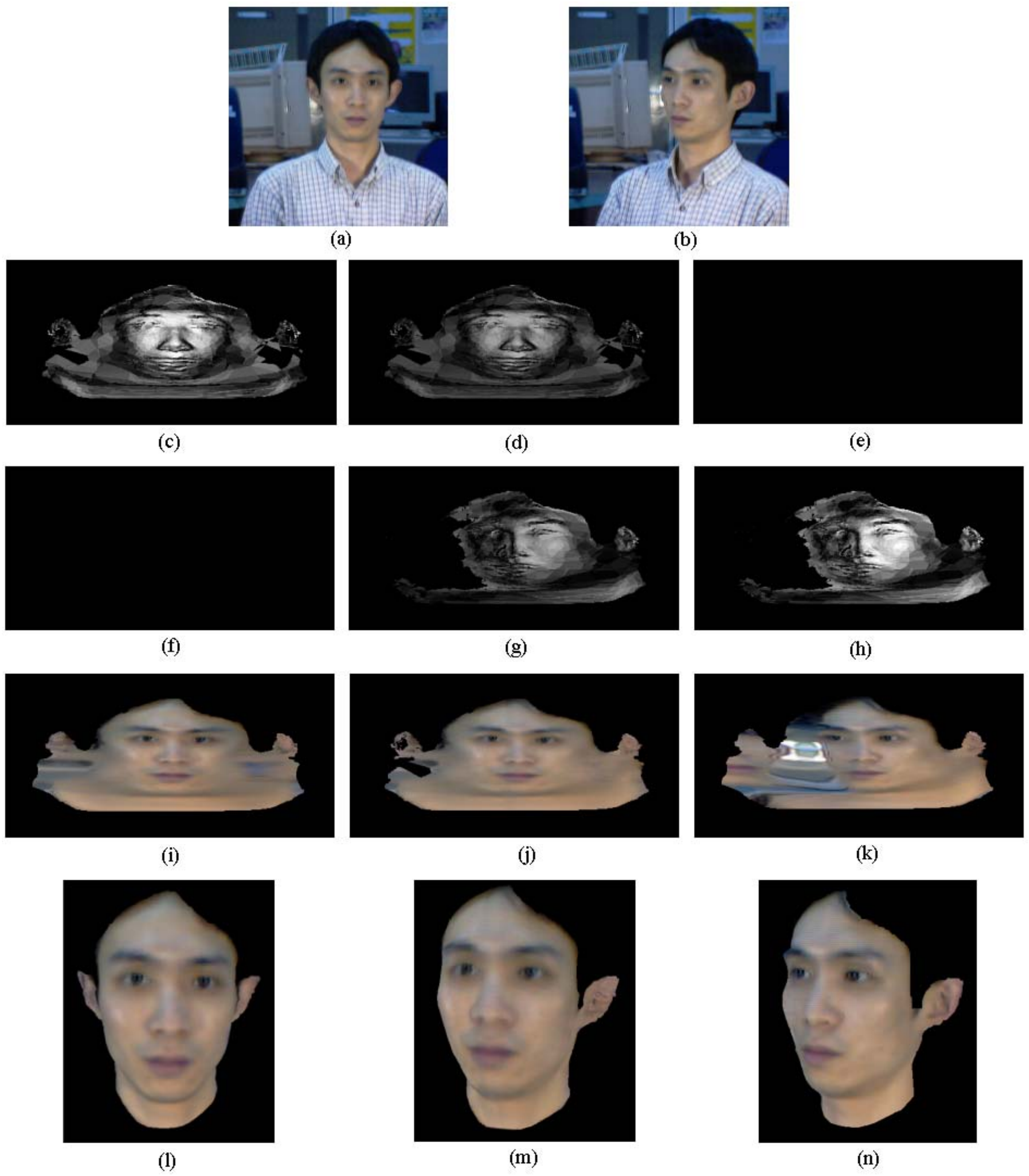


Figure 5: View-based texture extraction and mapping. Top to bottom: two camera views of the subject (a-b); the associated weight maps of (i) in different viewing directions (c-e); the associated weight maps of (k) in different viewing directions (f-h); cylindrical texture maps in which (i) and (k) are cylindrical projections of the facial model rendered by the actual views (a) and (b) respectively and (j) is the blended cylindrical texture map for rendering the facial model from the viewing direction of the virtual view; facial model rendered from different viewpoints in which (l) and (n) are rendered from two actual views and (m) is the result of the facial model rendered from the viewing direction of the virtual view.

After they are mapped onto the facial model from the recovered camera positions, the rendered facial model is projected to texture space, resulting in cylindrical texture maps as shown in Fig. 5 (i) and (k). Note that the left and right columns of the remaining images correspond to the viewing directions of actual views (a) and (b) respectively, and the middle column corresponds to the viewing direction of an in-between virtual view. In the second and third rows the weighting functions associated with Fig. 5 (i) and (k) at different viewpoints are shown. Fig. 5 (j) shows the blended texture map of the virtual view from Fig. 5 (i) and (k) using view-dependent weighting function. The bottom row shows the 3D facial model rendered from the viewing directions of actual views (Fig. 5 (l) and (n)) and final result of rendering face from the viewing direction of the virtual view (Fig. 5 (m)). The texture map that results from this process does a good job of smoothly rendering the facial model across different viewpoints in an automatic way. The areas where the extracted texture map does not provide reasonable results are the ears that have an intricate geometry with many folds and usually cannot be correctly scanned and registered with the images. Their irregular shape also makes them fail to project without overlap on a cylinder. One possible way to tackle this problem is to compute the texture maps for ears by projecting the corresponding mesh part onto a selected input image where the ears are clearly visible (e.g., side views).

5. Conclusion

In this work we have described the ***digitization process***, ***digitized objects*** and ***digitized textures***. We have also implemented the texture mapping process. There are several issues that still exist that need to be improved. In the ***digitization process***, lasers are line-of-sight dependent so there may be areas of an object that are hidden and unable to be scanned readily. There are ways to overcome this limitation both in hardware and software. Dark and transparent objects are a problem as laser light is absorbed more than reflected, which cannot be detected by the vivid700 scanner, there is also loss of data around regions where there are large variations of texture or curvature, such as places near the chin and neck. Overall the performance of vivid is quite satisfactory. In the case of ***digitized textures***, the texturing can also be the basis for many more sophisticated rendering algorithms for improving visual realism and quality. For example, environment mapping is a view-dependent texture mapping technique that supplies a specular reflection to the surface of objects. This makes it appear that the environment is reflected in the object. More generally texturing can be thought of as a method of providing (or perturbing) parameters to the shading equation such as the surface normal (bump mapping), or even the coordinates of the point being shaded (displacement mapping) based on a parameterization of the surface defined by the texture coordinates.

Acknowledgments

The authors would like to thank CGIT, CAMTech, SCE and NTU for all the support extended to carry out this research.

References

[1] Fabrice Neyret and Marie-Paule Cani, Pattern-Based Texturing Revisited, Proceedings of SIGGRAPH 99, 1999, ACM SIGGRAPH, Addison Wesley, Aug, pp. 235 - 242.

[2] P. V. Sander, J. Snyder *et al*., Texture Mapping Progressive Meshes, Computer Graphics, Proceedings of SIGGRAPH 2001.

[3] Mark Segal, Carl Korobkin *et al*., Fast shadows and lighting effects using texture mapping, Proceedings of the 19th annual conference on Computer graphics July 27 - 31, 1992, Chicago, IL USA, pp. 249-252.

[4] John F. S. Yau and Neil D. Duffy, A Texture Mapping Approach to 3-D Facial Image Synthesis, *Computer Graphics Forum*, 7(2), pp. 129-134, June 1988.

[5] J.F. Blinn and M.E. Newell, Texture and Reflection in Computer Generated Images, CACM, 19(10), October 1979, pp. 542-547.

[6] Takaaki Akimoto, Yasuhito Suenaga, and Richard S. Wallace, Automatic creation of 3D facial models, *IEEE Computer Graphics and Application,* 13(5): 16–22, September 1993.

[7] W. -S. Lee and N. Magnenat-Thalmann, Fast head modeling for animation, *Image and Vision Computing,* 18(4): 355–364, 2000.