HARDWARE-BASED TEXTURE EXTRACTION FOR BUILDING FAÇADES

Martin Kada

Institute for Photogrammetry (ifp), University of Stuttgart, Germany
Geschwister-Scholl-Strasse 24D, D-70174 Stuttgart
martin.kada@ifp.uni-stuttgart.de

Commission IV, WG 6

KEY WORDS: Extraction, Building, Texture, Visualization, Hardware, Graphics, Reconstruction

ABSTRACT:

The reconstruction of 3D city models has matured to the point where large data sets are now available. As most of the data collection methods used are based on airborne sensors like e.g. aerial laser scanning or stereo imagery, the detailed geometry and material of the building façades is typically not available. For visualisation purposes, however, the surface structure is essential to achieve a good visual impression of the respective buildings. An efficient technique to provide the missing building façades is to extract texture images from terrestrial photographs and map them to the polygonal faces of the reconstructed models. As the task of manual texture extraction and placement is very time-consuming, an automatic approach is presented in this article that utilises the rendering pipeline of modern 3D graphics cards. The problem of texture extraction can therefore be solved by using graphics algorithms that are nowadays implemented in hardware and consequently are extremely fast. Since only parts of the building or even of a façade are typically captured in one single image, self occlusions of the buildings are detected and several photographs taken from various positions are fused to generate the final texture images. In order to gain high quality textures, the lens distortion of calibrated cameras is corrected on-the-fly by the use of pixel shaders that are running on the programmable graphics processing unit.

1. INTRODUCTION

The acquisition of 3D city models has been of major interest for the past years and a number of algorithms are now available both for the automatic and semiautomatic collection of 3D building models. Based on measurement from aerial stereo imagery or airborne laser scanner data, the geometry of buildings can be reconstructed on a large scale. (Baltsavias, Grün and van Gool, 2001) e.g. give a good overview of experimental systems and commercial software packages. One major limitation of these approaches is, however, that the resulting models have rather coarse façades. (Früh and Zakhor, 2003) present a method that merges ground based and airborne laser scans and images. The additional terrestrial data naturally leads to more detailed façades.

Key market for this type of data is the visualisation in the context of city planning, three-dimensional car navigation, virtual tourism information systems and location based services. In addition to pre-rendered movies of the virtual environments where the user has no freedom of movement, real-time visualisation is getting more and more important. (Kada et al., 2003) show e.g. that literally a complete city can be interactively displayed in 3D on today's consumer PC systems (see Figure 1).

For the photo-realistic visualisation of urban landscapes, the material of the building façades is essential for the visual impression. An efficient technique to model building façades is to extract texture images and place them on the coarse, polygonal faces of the reconstructed models. Whereas roof images can easily be acquired from aerial photographs, such an approach is not feasible for the building façades. It is therefore inevitable to use terrestrial images as the source for high-quality façade textures. The manual texture extraction and placement is, however, a tedious task and can easily take up to several days per building for good results (see Figure 2). Such an approach is consequently not applicable for capturing a large number of building façades.

Figure 1. A 3D landscape model of Stuttgart rendered in a real-time visualisation environment. All façade textures of buildings located in the main pedestrian area were manually captured.

In this article, an approach is described that automatically extracts façade textures from terrestrial photographs and maps them on geo-referenced 3D building models. If the exterior orientation of the camera is known, a transformation can be computed that projects the polygonal faces of the building model into the image. (Klinec and Fritsch, 2003) determine the rotation and translation of the exterior orientation by searching for correspondences between object and image features and use them in a photogrammetric spatial resection.

(a) (b) (c) (d) (e)

Figure 2. The manual texture extraction and mapping process involves (a) digital capturing of the building façade and removal of lens distortions, (b) selection of quadrilateral image sections, (c) rectification of perspective distortions, (d) retouching of occluded areas and (e) texture placement on building model.

The presented texture extraction approach ties in from when the exterior orientation of the photographs is known. It describes an efficient way to get from the correspondences of 3D object and image points to a completely textured building model that is ready for visualisation purposes. A naive way to visualise this kind of data would be to define the corresponding points in the image as texture coordinates and use e.g. a VRML viewer for scene rendering. This is not a viable approach, however, as complex geometric image transformations can not be realised due to perspective effects or lens distortion.

By using the functionality of 3D graphics hardware, the whole process can be realised very efficiently so that interactive tools are possible. The user e.g. defines the input photographs and extraction parameters and observes the resulting textured building model in real-time. Any self occlusions of the model are detected automatically, so that several images can be fused to get the final texture images. The camera distortions are removed transparent to the operator in the hardware so that no extra care needs to be taken.

Because 3D APIs and SDKs nowadays provide powerful and functional rich interfaces, such a texture extraction system can be realised with very little programmable effort.

2. HARDWARE-BASED TEXTURE EXTRACTION

The texture extraction approach described in this article utilises new technologies that can be found in today's commodity 3D graphics hardware. Especially three developments are of greater importance and shall be briefly discussed in the following sections.

2.1 Graphics Processing Units

The extraction of façade textures from digital images mainly involves the transformation of vertices and the processing of pixel data, computations that can be highly parallelized for increased performance. Because the main CPU does not exploit this parallelism very effectively, a software solution is generally not an adequate approach. Graphics processing units (GPU) that are integrated in today's commodity PC graphics cards, however, are optimised for this kind of data processing. As graphics processors have evolved from a fixed function to a programmable pipeline design, they can now be utilised for various fields of applications.

2.2 Shader Languages

Shaders are small programs that are executed on the 3D graphics card. They can be conceived of as functions that are called within the GPU at specific points during the generation of the image. Two types of shaders exist: vertex shaders replace the transformation module in the geometry stage and pixel shaders replace the processing of individual pixels in the rasteriser stage of the graphics rendering pipeline.

Nowadays, shaders can be developed using High-Level Shader Language (HLSL developed by Microsoft) (Gray, 2003) or C for graphics (Cg developed by NVIDIA) (Fernando and Kilgard, 2003). Both are based on the programming language C and offer the flexibility and performance of an assembly language, but with the expressiveness and ease-of-use of a high-level language.

In the presented approach, pixel shaders are used to exert control over (projective) texture lookups, the depth buffer algorithm and to realise the on-the-fly removal of lens distortions for calibrated cameras.

2.3 Floating-Point Texture Format

Textures in floating-point format can hold real 32 bit colour values per channel. If used in combination with a pixel shader, a texture must not necessarily hold colour values, but rather all kinds of per pixel floating-point data can be stored in it. The pixel shader knows how to interpret the data in order to compute the output colour and depth values.

Through the use of standard 3D APIs like OpenGL or Direct3D, the extraction algorithm is just a matter of setting up the rendering pipeline and to provide the vector information of the building geometry. The complexity of the core algorithm amounts to only a few lines of code.

3. ALGORITHM

It is assumed that the building geometry is already available as a 3D geo-referenced, polygonal surface model. The input photographs were taken with a calibrated camera and their exterior orientations are known. Hence, the transformations that project the faces of the building model into the images

can be computed. Each projected polygon then overlays all the pixels that unprojected will make up the final façade texture (see Figure 3).

Figure 3. Projected 3D building model overlaid on the input photograph (Rosensteinmuseum).

Because the façade textures have to be represented by quadrilaterals, the polygons are substituted during the extraction process by their bounding rectangles which are given in three-dimensional world-space coordinates. The texture extraction is basically performed by rendering a quadrilateral with the colour values of the unprojected image pixels of the bounding rectangle.
Lens distortions are removed on-the-fly in the pixel shader, so the extraction works as if processing idealized images where the calibration parameters are applied.

3.1 Texture Extraction

The first step is to set up the rendering pipeline to fill the entire target buffer where the final façade texture will be rendered to. For this purpose, all transformation-related states (including the one for the projection) are initialised with the identity matrix. Drawing a three-dimensional unit square with vertices v_0 = (-1, 1, 0), v_1 = (1, 1, 0), v_2 = (1, -1, 0) and v_3 = (-1, -1, 0) will render all pixels in the target buffer as wanted. The four vertices can incidentally be thought of as the projected vertices of the polygon's bounding box into the image plane.
So far, however, the rasteriser would only render a blank façade image as no colour information is provided yet. Therefore, a photograph must be assigned to the pipeline as an input texture from where to take the colour information from. As mentioned above, the polygon's projected bounding box defines the pixels to be extracted from the input texture. So in addition to the above mentioned vertices, the texture coordinates of the four vertices are specified as the four-element (homogenous) world space coordinates of the bounding box. Setting the texture transformation matrix with the aforementioned transformation from world to image space concludes the initialisation.
During the rendering of the target façade image, the rasteriser linearly interpolates the four-dimensional texture coordinates across the quadrilateral. A perspective texture lookup results in the perspectively corrected façade texture (see Figure 4). Some extra care has to be taken, however, as the results of this transformation are in the range -1 to 1 and the final texture coordinates are indexed in the range 0 to 1. A single scale and bias will map the coordinates accordingly.

Figure 4. Extracted façade textures.

The resulting façade texture can then be read from the frame buffer and saved to file. Most 3D APIs provide special functions for this task.

Figure 5. The 3D building model with the extracted textures placed on the façade polygons.

3.2 Texture Placement

After the extraction, the textures need to be placed on the corresponding polygons (see Figure 5). In order to find the two-dimensional texture coordinates for the polygon vertices, a function identical to glTexGen (Shreiner, 2003) of OpenGL is used. The function automatically generates texture coordinates s and t by a linear combination of the vertex coordinates:

$$\begin{aligned} s &= A_1 x + B_1 y + C_1 z + D_1 \\ t &= A_2 x + B_2 y + C_2 z + D_2 \end{aligned} \qquad (1)$$

A, B, C and D can be thought of as the definition of planes in parameter form. The normal vector components A, B and C of the two planes are defined by the vector from the bottom left vertex to the bottom right vertex of the bounding box and from the bottom left vertex to the top left vertex accordingly. The values for D are simply computed by inserting the bottom left vertex into the equation. The result of the linear

combination of the polygon vertices are then in the range 0 to 1 as required.

3.3 Detection of Self-Occlusions

The extraction of façade textures from photographs always leads to the problem that parts of the facades are not visible because of self-occlusions of the building. If no special care is taken, then erroneous pixel values are extracted for the occluded parts of the façade (see Figure 6 to Figure 8). To avoid such artefacts, invalid pixels that belong to other polygons must be identified and marked.

Pixel-wise occlusion detection is realised in this approach by using the depth buffer algorithm. First, the depth value of the closest polygon is determined for each pixel in the photograph and stored in a depth texture. This can simply be done by rendering all polygons with the hardware depth buffer functionality enabled and by copying the resulting depth buffer into a 32 bit floating-point texture. A more efficient approach is to calculate the depth value in a pixel shader and render directly into the depth texture. Modern 3D graphics processors support the floating-point texture formats even as render targets.

During texture extraction, the depth value is read out in the pixel shader using the same texture coordinates as for the colour lookup. After the perspective divide is applied to the texture coordinates, the z-component holds the depth value for the current polygon. A comparison of these two depth values then determines if the pixel in the colour texture belongs to the polygon. If e.g. the value from the depth texture is lower then the computed value, then the polygon is occluded at this pixel by another polygon. Figure 9 shows some results where occluded pixel values have been blackened out. To suppress artefacts caused by precision errors, the depth test is done by applying a small depth bias in the depth test.

3.4 Image Fusion

As only parts of a building or even of a façade are typically captured in one single image, the colour information from several photographs that were taken from various positions need to be combined to generate the final façade textures. One simple approach is to extract several textures for the same polygon from all available photographs and then use the one with the fewest pixels marked as occluded (see e.g. Figure 10). Other criteria may possibly be the photograph taken closest to the façade or the one with the best viewing angle.

The problem of image fusion done in the hardware is how to get the graphics pipeline to decide from which image a pixel should be taken. The solution is to process all images and make the hardware accept or reject pixel values by using the depth, stencil or alpha test. Even though the approach is brute force, it is still very efficient with the hardware support.

The presented per pixel approach merges the final façade texture by using the colour value of only the closest, non-occluded pixel found in all images. The occlusion detection works as described in the previous section, but now with the depth test enabled on the hardware. The output of the pixel shader that is used for the hardware depth buffer test is the calculated depth value for non-occluded pixels and 1.0 (the farthest possible depth value) for occluded pixels. Because it is usually better to have a wrong colour value rather then no

Figure 6. Input photograph showing Stuttgart State Theatre.

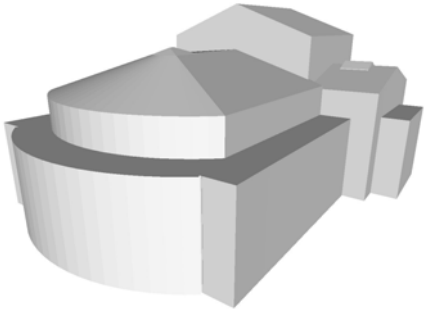


Figure 7. 3D building model without textures.

Figure 8. Building model automatically textured without occlusion culling.

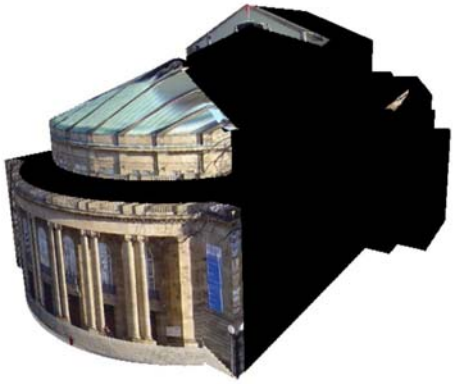


Figure 9. Building model automatically textured with occlusion culling. The black pixels were marked as occluded texture pixels.

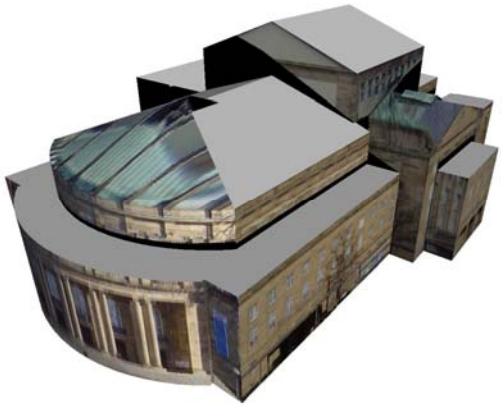
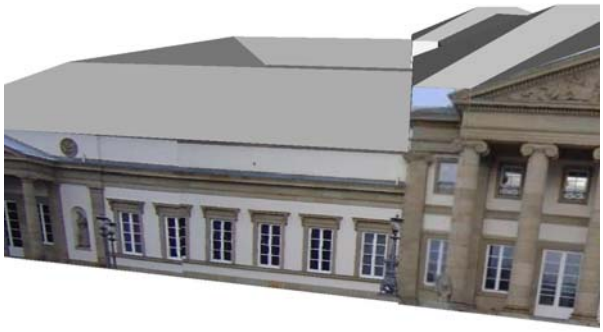


Figure 10 The resulting textures with the fewest occluded pixels were chosen as the façade texture and placed on the building model.

colour value, the pixel colour is always written. The hardware depth test ensures that always the closest pixel is taken and the artificial depth value of 1.0 gives precedence to non-occluded pixels. The result can be seen in Figure 11.

(a)



(b)

Figure 11 (a) Façade texture extracted from two input photographs using per-pixel fusion. The images were taken from two different positions. (b) The extracted façade textures placed on the building model.

3.5 Removal of Lens Distortion

If a calibrated camera is being used to capture the building facades, the lens distortion in the images are corrected on-the-fly in the pixel shader. The major benefit of this approach is that the extraction process works with the original images. Image pixels are therefore filtered only once during the whole process as opposed to the alternative approach where the idealised image is computed beforehand. The result is that the extracted façade textures are of higher quality.

The lens distortion is described by the parameter set introduced by (Brown, 1971) and denotes the transition of pixels from the distorted to the idealized image. Here, the following subset is used as in (Fraser, 1997):

$$\begin{aligned} \Delta x &= \Delta x_i + \Delta x_r + \Delta x_d + \Delta x_a \\ \Delta y &= \Delta y_i + \Delta y_r + \Delta x_d + \Delta y_a \end{aligned} \tag{2}$$

Change of interior orientation:

$$\begin{aligned} \Delta x_i &= \Delta x_0 - \frac{\overline{x}}{c}\Delta c \\ \Delta y_i &= \Delta y_0 - \frac{\overline{y}}{c}\Delta c \end{aligned} \tag{3}$$

Radial distortion:

$$\begin{aligned} \Delta x_i &= \overline{x}(r^2 K_1 + r^4 K_2 + r^6 K_3) \\ \Delta y_i &= \overline{y}(r^2 K_1 + r^4 K_2 + r^6 K_3) \end{aligned} \tag{4}$$

Decentring distortion:

$$\begin{aligned} \Delta x_d &= (r^2 + 2\overline{x}^2)P_1 + 2\overline{xy}P_2 \\ \Delta y_d &= 2\overline{xy}P_1 + (r^2 + 2\overline{y}^2)P_2 \end{aligned} \tag{5}$$

Affinity and shearing:

$$\begin{aligned} \Delta x_a &= \overline{x}B_1 + \overline{y}B_2 \\ \Delta y_a &= 0 \end{aligned} \tag{6}$$

Because the extraction process needs the transition of pixels from the idealized to the distorted image and the formula is not invertible, an iterative method is be used. Unfortunately, arbitrary iterations are not supported by current 3D graphics hardware. But because the graphics API Direct3D 9.0 (Microsoft, 2003) already defines dynamic flow control in Pixel Shader 3.0, graphics cards are likely to include this feature in the near future. The removal of lens distortion can at that time be completely computed in hardware.
Until then, an alternative approach must be used. The 2D transition vectors are pre-computed for all pixels in the image and stored in a two times 32 bit floating-point texture. In the pixel shader, a first texture look-up gets the transition vector and adds the correction values to the texture coordinates. The new coordinates are then used for the depth and colour lookup.

4. IMPLEMENTATION AND RESULTS

The design goal of the implementation was to have a vendor independent system, which means that the algorithms should work with a broad variety of 3D graphics cards. The graphics API of choice was therefore Direct3D 9.0, which also includes the high-level shader language (HLSL).

The calibration of the camera was done using the bundle program Australis (Fraser, 1997). Camera calibration files are automatically loaded and the correction textures are generated as needed. The input photographs were taken at resolution 1280 * 960 pixels, but resolutions up to 2048 * 2048 are also supported. This limitation comes from the fact that textures in the graphics hardware are limited to this size. The output resolution of the façade textures is at this point fixed at 256 * 256 pixels.
The performance analysis has been conducted on a standard PC with an Intel 4 3.0 GHz Processor, 1 GB of DDR-RAM and a graphics card that is based on the ATI 9800 GPU with 256 MB of graphics memory. The test results are given in Table 1. It should be noted that 8 or 16 input photographs for per pixel image fusion is not practical. This rather high number was solely used to show the speed of the approach. Nevertheless, the extraction time with all features enable is still below one second.

# images	Extraction Process	time	
		model A	model B
1		31 ms	47 ms
1	Lens Correction	32 ms	62 ms
1	Occlusion Detection	32 ms	63 ms
1	Occlusion D. + Lens C.	46 ms	78 ms
8	Image Fusion (per Pixel)	172 ms	328 ms
8	Image Fusion + Lens C.	203 ms	391 ms
16	Image Fusion (per Pixel)	375 ms	813 ms
16	Image Fusion + Lens C.	422 ms	829 ms

Table 1. Extraction times measured for model A (Rosensteinmuseum, 71 polygons) and model B (Stuttgart State Theatre, 149 polygons).

5. CONCLUSION AND FUTURE WORK

This article described the concept and the implementation for hardware-based texture extraction of photo-realistic façade textures. The implementation of such a system is shown to be very simple by using standard 3D APIs and shader languages. Fast extraction is possible on commodity PC hardware equipped with a 3D graphics processing unit and the resulting façade textures proved to be of very high quality. The resulting building models are automatically mapped by perspectively correct textures and can therefore be used for real-time visualisation.
As the system has a low response time, it has the potential to be extended towards a semi-automatic tool, which allows the refinement of the model based on manual measurement in terrestrial images. The manual fitting of available building geometry to terrestrial images is often required due to remaining errors in the building model. Such errors are of nuisance when the correspondence between object and image is not exactly given and lead to artefacts or even wrong façade textures. Hardware-based texture extraction will allow a real-time visualisation of the textured 3D model, so that the operator can immediately observe the geometric changes.
The future work will be to speed-up the overall process by doing some pre-processing of the geometry on the main CPU. Backface culling could e.g. be pre-computed for each image and stored in a backface table. Not all images would need to be processed for each polygon anymore. Another area of improvement is the quality for per pixel texture fusion. Alpha blending might help to reduce artefacts if parts of the texture can not be aligned correctly because of errors in the exterior orientation. As a combination of per-polygon and per-pixel image fusion promises the best results, adapted algorithms shall further be developed.
In order to address occlusions by other objects, the presented system could be extended to a semi-automatic tool where the operator marks pixels or regions in the photograph as invalid. These pixels will not be used in the final texture, but rather colour values from other photographs are used or the missing pixel colours are reproduced by subsampling algorithms.

6. ACKNOWLEDGEMENTS

The research described in this paper is founded by "Deutsche Forschungsgemeinschaft" (DFG – German Research Foundation). The research takes place within the Center of Excellence No. 627 "NEXUS – SPATIAL WORLD MODELS FOR MOBILE CONTEXT-AWARE APPLICATIONS" at University of Stuttgart. The geometry of the building models is provided by Stadtmessungsamt Stuttgart.

7. REFERENCES

Baltsavias, E. Grün, A. and van Gool, L., 2001. *Automatic Extraction of Man-Made Objects from Aerial and Space Images (III)*. Swets & Zeitlinger B.V., Lisse, The Netherlands.

Brown, D.C., 1971. Close-Range Camera Calibration. *Photogrammetric Engineering*, 37 (8), pp. 855-866.

Fernando, R. and Kilgard, M., 2003. *The Cg Tutorial*. Addison-Wesley.

Fraser, C.S., 1997. Digital Camera Self-Calibration. *ISPRS Journal of Photogrammetry and Remote Sensing*, Vol. 52, pp. 149-159.

Früh, C. and Zakhor, A., 2003. Constructing 3D City Models by Merging Aerial and Ground Views. *IEEE Computer Graphics and Applications*, Vol. 23 No. 6, pp. 52-61.

Gray, K., 2003. *The Microsoft DirectX 9 Programmable Graphics Pipeline*. Microsoft Press.

Kada, M., Roettger, S., Weiss, K., Ertl, T. and Fritsch, D., 2003. Real-Time Visualisation of Urban Landscapes Using Open-Source Software In: *Proceedings of the ACRS 2003 ISRS*, 24th Asian Conference on Remote Sensing & 2003 International Symposium on Remote Sensing, Busan, Korea. (On CD-ROM)

Klinec, D. and Fritsch, D., 2003. Towards Pedestrian Navigation and Orientation. In: *Proceedings of the 7th South East Asian Survey Congress*, SEASC'03, Hong Kong. (On CD-ROM)

Microsoft, 2003. DirectX Documentation for C++. *Microsoft DirectX 9.0 SDK*. http://msdn.microsoft.com/library/default.asp?url=/downloads/list/directx.asp

Shreiner, D., Woo, M. and Neider, J., 2003. *OpenGL Programming Guide (Version 1.4)*, Addison-Wesley.