Profiling

Appendix A

A.1 Generating view-specific textures

Chapter 5 described two algorithms for back-projecting the reference images onto the hypothesised surface. The first algorithm relied only on OpenGL's fixed-pipe functionality, but had to overcome the graphics hardware's clipping limitations by pre-processing the scene-graph on the CPU. In contrast, the programmable pipe version has better scope for manipulating vertex attributes within the graphics hardware and therefore does not require pre-processing. The disadvantage of this approach is an increased overhead in the graphics pipe to combine points in world and texture spaces into a single OpenGL vertex, and a more involved back-face culling process that is compelled to test each fragment rather than culling an entire triangle at once.

The fixed-pipe algorithm requires significantly more pre-processing for each instance of the hypothesised geometry, while the programmable pipe-line algorithm is able to compile the geometry into display lists on the graphics hardware. In contrast to the programmable pipe's approach—where all triangles are sent to the graphics pipe-line—the fixed-pipe algorithm's pre-processing step can eliminate a large number of back-facing triangles from further consideration. Consequently, the fixed-pipe algorithm generates less spurious fragments (ie. ones that do not contribute to the texture) and is therefore an advantage to graphics hardware with slow fragment processors.

We profiled the execution time of both the fixed-pipe and programmable-pipe algorithms in a variety of scene-configurations to determine the relative benefits of moving the scene pre-processing from the CPU into the vertex-shader. The algorithms' execution time is a function of four quantities:

- the total number of surface triangles;
- the number of triangles that face toward a reference camera;

- the number of front-facing triangles that must be clipped against the reference camera's imaging plane; and
- the number of fragments generated by the rasteriser.

The experiments were designed to test the trade-off of pre-processing in software versus in-hardware processing by varying the number of input triangles; the number of front-facing triangles; and the number of triangles that are clipped by the imaging plane. Each test measured the time taken to back-project a single reference image onto the hypothesised surface; the total execution time including the process of accumulating *all* view-specific textures and measuring their consistency is left for the next section. In each test, a reference image was back-projected onto a given hypothesised scene configuration 1000 times; the performance is described in terms of the number of view-specific textures generated per second. The tests were run on a machine with an Intel Core 2 Quad CPU clocked at 2.4GHz and an *n*Vidia GeForce 8800GT clocked at 600Mhz with memory bandwidth of 57.6 GB/second. Although the fill rate is an important factor in both algorithm's performance, only 512×512 textures were generated. As will be shown later, neither algorithm is fill-rate limited, and therefore fill-rate will only affect scale and not relative performance.

A.1.1 Triangle scalability

The overhead related to the algorithms' per-triangle processing was tested by back-projecting a reference image onto a tessellated mesh. The mesh is built by dividing a plane into n triangle strips, where strip comprised $2n$ triangles. All triangle strips were front-facing so that both algorithms were compelled to send the same number of vertices to the graphics pipe, although the fixed-pipe algorithm was still required to check the orientation of each triangle. The test configuration is illustrated in Figure A.1(a). The time taken to back-project the reference image onto the surface was measured over 1000 iterations and illustrated in Figure A.2 against the number triangle strips in the hypothesised surface.

The results show that the programmable pipe-line version is more efficient than the fixed-pipe version for a sufficiently complicated scene, but not as efficient for scenes with less than $4(4 \times 2)$ triangles. The 32 triangle barrier seems to indicate the saturation point where both presentation, vertex and fragment shaders units are full. The graph suggests that the load balance of the fixed-pipe is better until this point by occupying all three stages. The programmable pipe, in contrast, hardly uses the CPU to present the

data to the pipe since the algorithm is able to use compiled geometry; it is only until the fragment processor is saturated that the overhead incurred by the increased per-vertex processing is mitigated.

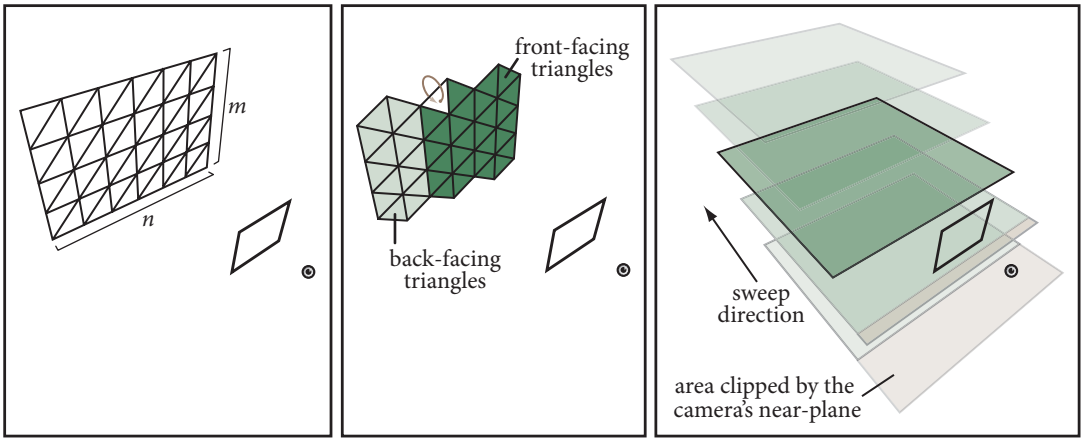
A.1.2 Back-facing culling

The back-face culling test determines the advantage of removing back-face triangles before presenting the scene to the graphics hardware. Figure A.1(b) illustrates the scene configuration, comprising 5,000 triangles that projected entirely within the image of the reference camera. Figure A.3 graphs the texture rate with an increasing proportion of them facing backwards with respect to the reference camera. As expected, the fixed-pipe's rate increases monotonically because backwards facing triangles are not sent to the graphics pipe. In contrast, the programmable pipe is unaffected by the number of backwards facing triangles because the back-facing test is applied in the fragment processor and fragments are textured irrespective of their orientation.

A.1.3 Image-plane clipping

The programmable pipe-line is able to use the OpenGL's near plane to clip surface triangles against the reference camera's imaging plane, unlike the fixed pipe-line, which must clip the triangles on the CPU. The overhead associated with modifying the geometry stream was tested by incrementally moving a stack of 1000 quadrilaterals towards a reference camera. As illustrated in Figure A.1(c), the stack was iteratively moved towards the camera along the reference camera's optical axis until the stack was entirely behind the reference camera's imaging plane. The stack's position denotes three clipping states: firstly, that the stack is not clipped because it is entirely in front of the camera; secondly, the stack crosses the imaging plane and is therefore subject to clipping; and, thirdly, that the stack is entirely behind the camera and is therefore not rendered.

The average texture rate as the stack is translated towards the reference camera is illustrated in Figure A.4. The results demonstrate that the performance of the fixed pipe-line algorithm is significantly affected by clipping, unlike the performance of the programmable pipe-line. The fixed pipe-line clearly indicates three levels of performance, corresponding to events when the triangle stack begins to be clipped by the reference camera's imaging plane at $z = -1$ and when the stack is entirely behind



(a) Resolution test. **(b)** Back-face culling test. **(c)** Clipping test.

Figure A.1: The three scenes designed to test the relative performance of the fixed- and programmable-pipe back-projection algorithms.

the imaging plane at $z = 1$. As expected, the programmable pipe-line's performance is unaffected by clipping because all triangles are sent to the graphics pipe.

A.1.4 Conclusion

Figure A.2 illustrates that the programmable-pipe has a scales better than the fixed-pipe algorithm. Although the programmable pipe must raster more fragments than the fixed pipe and—as illustrated by Figures A.3 and A.4—is unaffected by clipping or removing triangles from the input stream, it is consistently more efficient than the fixed-pipe algorithm for a sufficient number of triangles. The constant texture rate in Figures A.3 and A.4 in the case of the programmable pipe algorithm, and—after accounting for the change in clipping state—in Figure A.4 for the fixed pipe-line, indicates that neither algorithm is fill-rate limited. Accordingly, the change in texture resolution would affect only the absolute performance of the two algorithms.

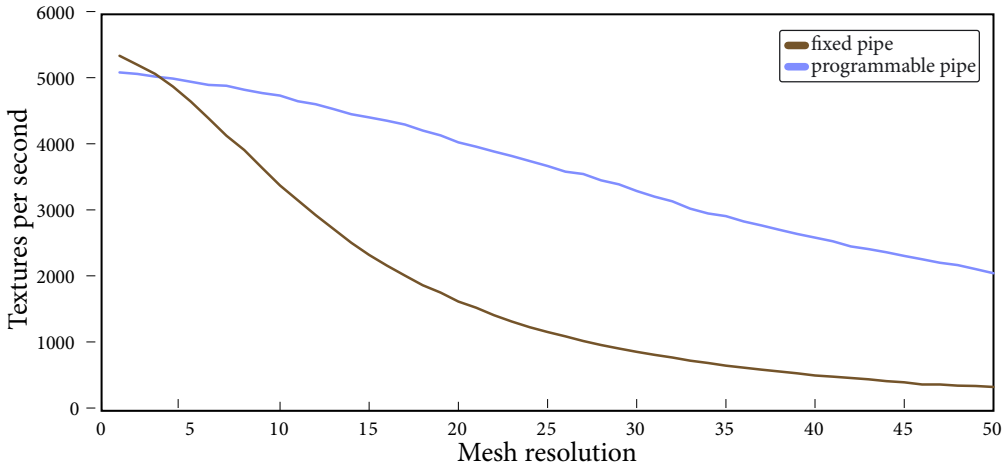


Figure A.2: The texture rate with respect to an increasing number of triangles.

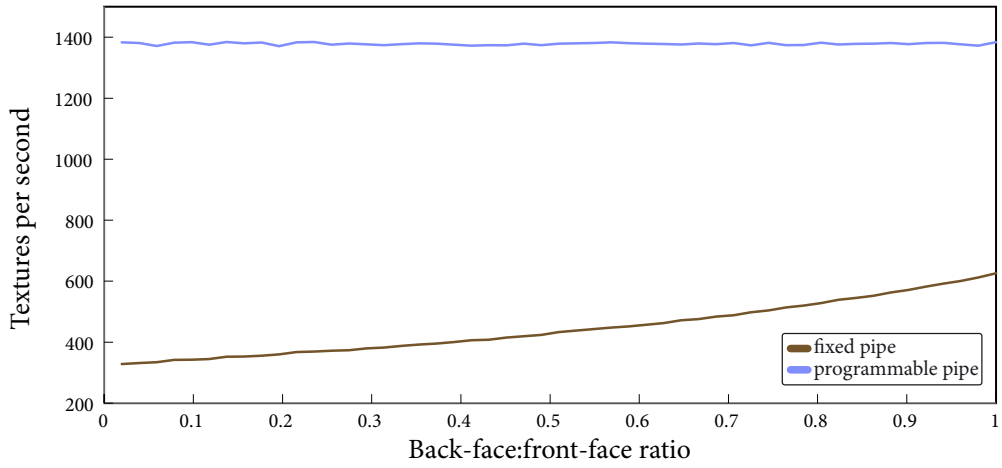


Figure A.3: The texture rate with respect to an increasing proportion of back-facing triangles.

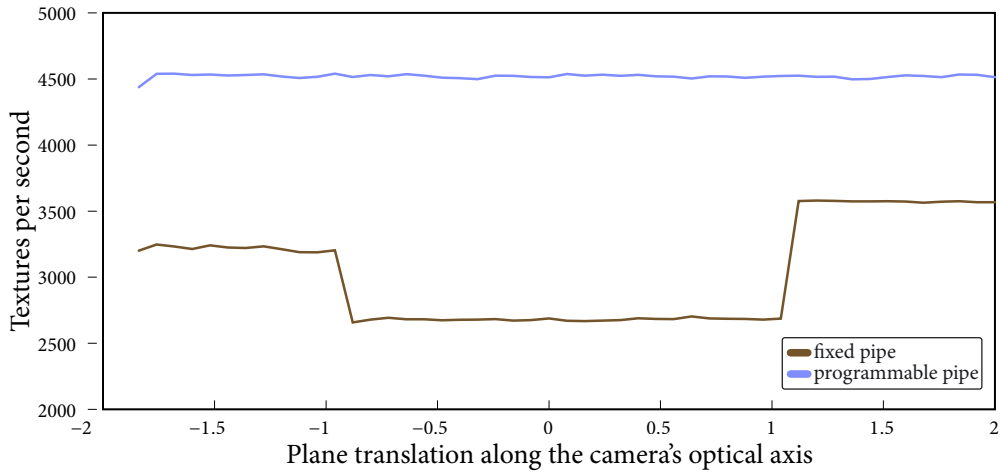


Figure A.4: The texture rate with respect to incrementally clipped triangle stack.

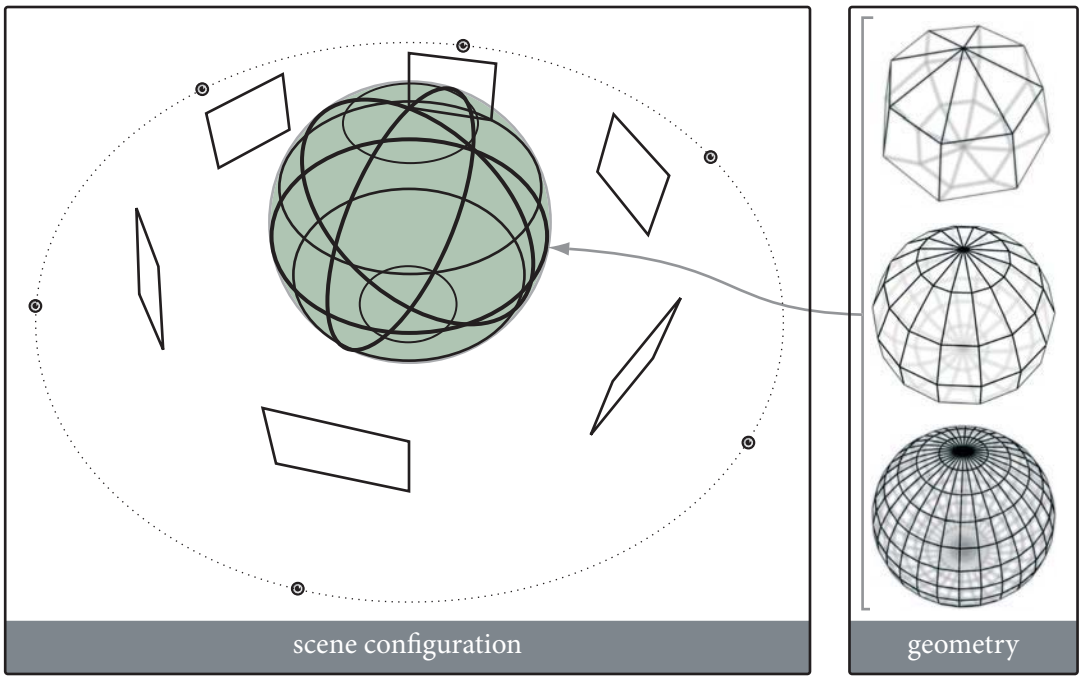


Figure A.5: The scene configuration for evaluating the texture rate with respect to variable number of cameras and surface complexity.

A.2 Hypothesis evaluation rate

The performance results in Section A.1 only considers the problem of generating a view-specific texture with respect to a single reference camera. Evaluating the surface-based likelihood, however, requires generating a view-specific texture for each reference camera, combining the textures into a composite texture, and evaluating the consistency of all view-specific textures. A key component of the surface-based likelihood involves identifying interior surfaces. The complexity of the algorithm described in Section 7.1.2 is squared with respect to the number of surface triangles. The image-based likelihood, on the other hand, back-projects one reference image onto the hypothesised scene and renders the result via a second reference camera. Because all image pairs must be considered, the complexity of the image-based likelihood is squared in the number of reference cameras.

The performance of both the surface-based and image-based likelihoods was measured with respect to the scene complexity and number of reference cameras. This test involved the scene configuration illustrated in Figure A.5, where a ring

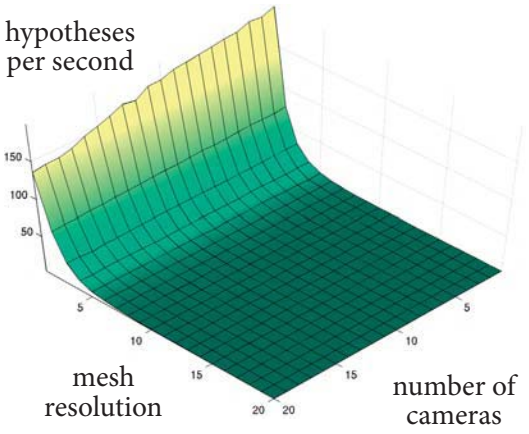
of reference cameras observed two concentric spheres. Because the surface-based likelihood assumes that primitives are not self-intersecting, two spheres used to ensure that the interior test must consider the intersection of two surfaces. The number of triangles comprising each sphere is given by $m = 8r(4 \times r + 1)$, where r is the sphere's resolution.

The timing results were generated by evaluating an hypothesised configuration over 1000 iterations using the surface-based and image-based likelihoods. Each evaluation is executed to the point of returning the scalar likelihood for each hypothesised surface, and the results are reported in terms of number of hypothesises evaluated per second. The programmable pipe-line algorithm was used to generate the surface-based likelihood. The three variations of likelihood functions used in the timing tests were:

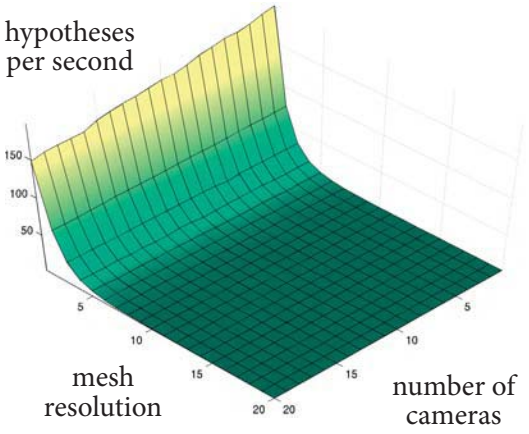
- **Figure A.6(a)**—the surface-based likelihood with variable weight composition;
- **Figure A.6(b)**—the surface-based likelihood with constant weight composition; and
- **Figure A.6(d)**–the image-based likelihood.

Figures A.6(a) and A.6(b) illustrate that the surface-based likelihood's performance is, as expected, dominated by the number of triangles in the hypothesised surface and linear with respect to the number of reference cameras. In contrast, Figure A.6(d) illustrates that the image-based likelihood's performance is primarily affected by the number of reference cameras.

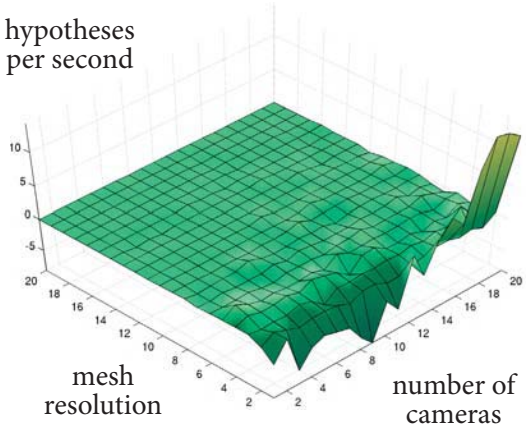
The variable weight composition, described in Section 6.3, has a different compositing path to that of simply averaging the visible regions over the set of view-specific textures. The difference in performance between the variable and fixed composition paths is illustrated in Figure A.6(c). The results are, in general, what we expected: the variable-weight implementation is slower than the constant-weight implementation, but that this penalty is amortized as the surface's complexity increases. Interestingly, these results show that the constant-weight implementation is *slower* for a large number of cameras and few triangles. This result is most likely to be an anomaly when the system was particularly busy when testing the constant-weight implementation, thereby skewing the results for this data subset.



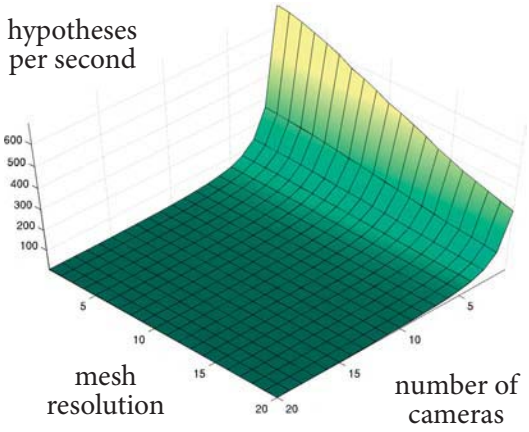
(a) Surface-based likelihood with variable composition.



(b) Surface-based likelihood with constant composition.



(c) The variable-weight overhead.



(d) The image-based likelihood.

Figure A.6: The texture rates of the surface-based and image-based likelihoods.

The Graphics Pipeline

Appendix B

Graphics hardware is very adept at generating synthetic images from a surface described by a collection of triangles and operations upon them. In the pursuit of increasing visual realism, the performance of graphics processors has rapidly improved to the point where it easily outstrips the peak theoretical performance of many general-purpose processors [40]. This dramatic increase in performance can be largely attributed to its architecture as a very specialised pipe-lined SIMD machine.

Recognising that graphics hardware is an exceptionally powerful computational resource, one of the the key motivations behind the development of the image- and surface-likelihoods was to perform as much of their computation on graphics hardware as possible. Although recent advances in graphics hardware continues to relax much of its rigid pipe-line model, however, it is far from being a completely general processor. Accordingly, the likelihoods had to be adapted to present the data in a manner congruent to the graphics pipe. This Appendix describes the order of operations and the limitations of the graphics pipe-line to give insight into some of the design decisions underpinning the computation of the photo-consistency likelihoods.

B.1 OpenGL

OpenGL is an API for graphics hardware concerned with rendering planar facets into a frame-buffer [50]. These facets are defined by a set of vertices which can be assembled into points, lines, triangles, quadrilaterals, and convex polygons. It does not have a mechanism for describing complex geometric objects, such as implicit surfaces. These higher-level surfaces must instead be approximated by a triangular mesh.

A set of attributes is associated with each vertex in the graphics pipe-line. Vertices are defined by setting graphics state-variables before 'binding' a vertex entity when a vertex position is admitted. A vertex necessarily describes a point in the local co-

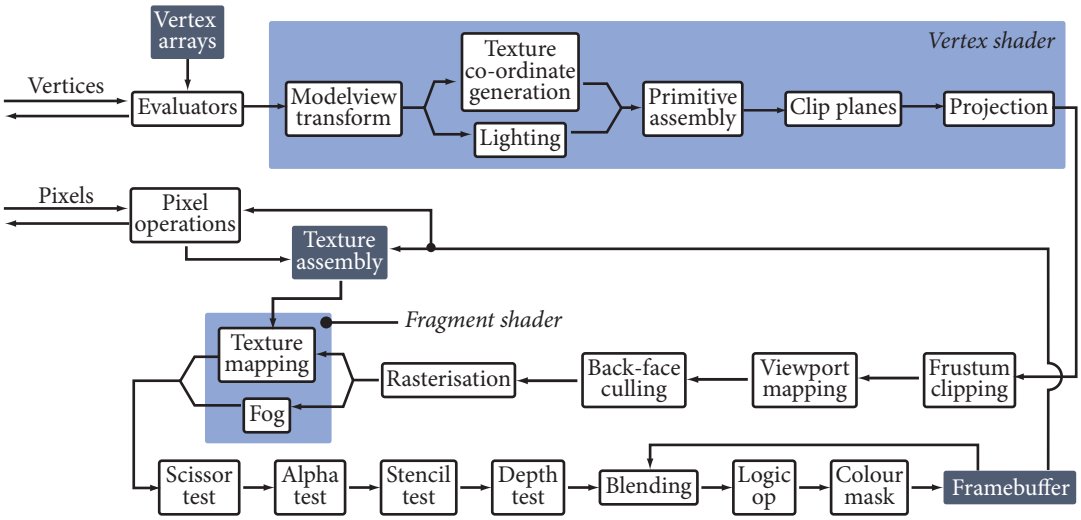


Figure B.1: The OpenGL pipe.

ordinate system, but may also include other attributes such as a normal vector, texture co-ordinates into multiple texture images, and a primary (and secondary) colour.

A stream of vertices are connected into a stream of facets. This geometry stream is subject to a number of operations, including transformations, lighting, texturing, blending, and pixel updates. The OpenGL's pipe-line abstraction describes the order of these graphics operations. The pipe-line is essentially a state-machine which is configured by changing the state of its many variables through the GL API. These variables affect the operation of the pipe-line itself, including defining the projection system, the texture to apply to the vertices, and a manner by which the rasterised triangles are blended with the frame-buffer. The abstract OpenGL pipe-line is illustrated in Figure B.1 and is composed of three key stages:

- *the vertex processor*, which can manipulates per-vertex attributes, such as transforming and lighting;
- *the fragment processor*, which computes colour for points on the surface; and,
- *frame-buffer operations*, which combines fragments with pixels in the frame-buffer.

The remainder of this Section describes the operation of the *fixed* pipe-line. Since OpenGL 2.0, the vertex and fragment processors have been programmable; small programmes, or *shaders* can be executed per-vertex and per-fragment for more exacting

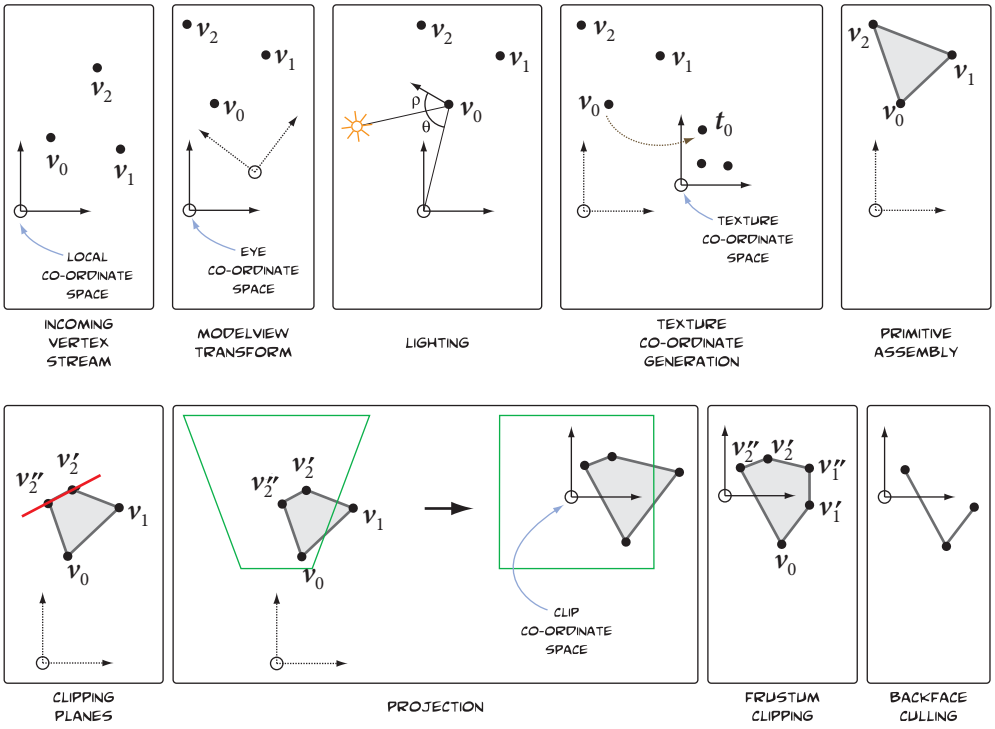


Figure B.2: The OpenGL vertex processor.

control over geometry attributes, although the input and output of the shaders is controlled by non-programmable units.

B.1.1 The vertex processor

Vertices can be sent to the graphics pipe-line by the client machine, retrieved from the vertex array or fabricated by NURBS evaluators. Either way, the input to the graphics pipe is a stream of incoming vertices describing primitives such as points, lines and triangles. The vertex processor maps vertices from their local co-ordinate space into image (or 'clip') co-ordinates and perform computation on the vertex attributes prior to rasterisation. Each incoming vertex is transformed by the 4×4 'model-view matrix'. This maps vertices into the camera (or 'eye') space where the imaging plane is aligned with the x/y axis. View-dependent lighting is applied to vertices in this space. Texture co-ordinates can also be modified by the texture matrix, or generated as a function of the vertices' position in eye-space.

After transformation, vertices are assembled into primitives which can be clipped

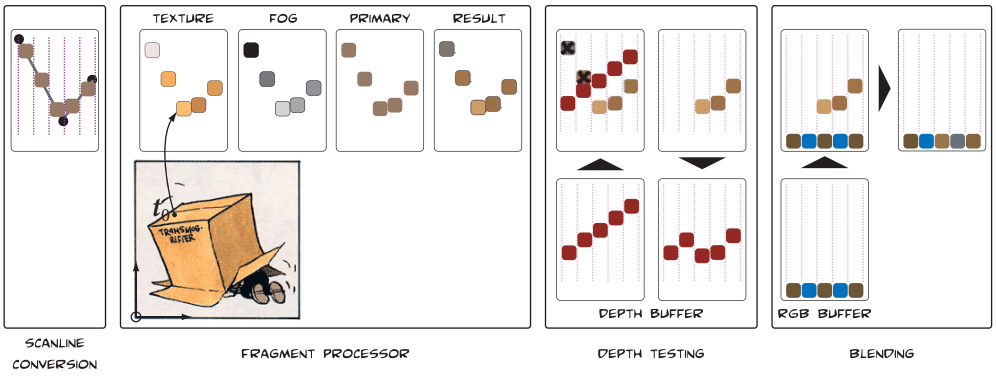


Figure B.3: The OpenGL fragment+frame-buffer engine.

against the arbitrary planes in eye-space. Depending on the primitive's type, this may fabricate new vertices along the clipping planes; new vertex attributes are generated by linear interpolation in this case. Vertices are then projected by a 4×4 projection matrix into the clip co-ordinate space.

The clip co-ordinate space maps the space visible by the camera into a cube bound by $x = \pm 1$, $y = \pm 1$ and $z = \pm 1$, representing the camera's *frustum* in clip-space. Primitives are clipped against this cube which may, again, involve fabricating new vertices through linear interpolation. Primitives that are within the cube and are not facing away from the camera are mapped to device co-ordinates, the space that corresponds to the actual frame-buffer rather than an abstract imaging space defined by the normalised clip-cube.

B.1.2 The fragment processor

With the primitive in device co-ordinates, the raster engine converts the projected facet into a stream of *fragments*. A fragment corresponds to a point sample in the primitive's image space. At least one fragment is generated per pixel, although some anti-aliasing approaches generate multiple samples per pixel. Associated with each fragment is a set of attributes—such as position, colour and texture co-ordinates—generated by linearly interpolating the facet's vertex attributes.

The fragment processor is responsible for computing the fragment's final colour from its attributes. The fragment's texture co-ordinates are used to find the point, or *texel*, in the current texture stored in the texture assembly. The fragment's texel is combined with its base colour (a linear interpolation of the lit primary colour defined

for each vertex) by the *texture environment*. The texel application can blend or replace the fragment's base colour with the texel's colour. Fog can also be applied by modulating the fragment's colour as a function of the distance from the image plane.

B.1.3 Frame-buffer operations

Fragments leave the fragment processor as coloured point-samples and are subject to a series of tests before being written to the frame-buffer. These per-fragment tests can remove fragments based on their attributes or by testing the fragment's attributes against the corresponding attribute stored in the frame-buffer. Each pixel in the frame-buffer stores a number of attributes synonymous with many of the fragment's attributes. The set of pixel attributes is often described as being a separate buffer; the 'depth-buffer,' for example, stores the depth of the fragment last written to it.

Fragments are tested by a series of filters; only fragments that pass all tests are written to the frame-buffer. It comprises two tests that can reject a fragment based solely on its attributes: the *scissor test*, which accepts fragments that fall within a rectangular region of the frame-buffer; and the α *test*, which accepts fragments whose α-channel passes a Boolean condition.

Fragments are also subject to tests against the frame-buffer attributes associated with the fragment's position in the stencil- and depth-buffers. The stencil buffer is an integer per pixel that can be incremented or decremented, depending on whether the fragment succeeds or fails the stencil test. For example, the stencil test may reject fragments if the corresponding stencil pixel is zero, but increment the stencil value in this case. The depth-test is used to reject fragments that fail a comparison between the fragment's depth and its corresponding position in the depth-buffer. The test typically rejects fragments that are further that previously rendered fragments, but other Boolean tests for equality can also be used.

The fragments that are not deleted are written into the frame-buffer. This may involve blending the fragment's colour with the frame-buffer's existing colour, replacing the corresponding value in the depth-buffer and adjusting the stencil value. The blend equation is of the form,

$$d_{rgb} = f_s s_{rgb} \circledast f_d d_{rgb} \tag{B.1}$$

$$d_\alpha = f'_s s_\alpha \circledast f'_d d_\alpha \tag{B.2}$$

where d and s are the *source* (the fragment) and *destination* (the frame-buffer) colours respectively. The coefficients f_s and f_d control the weights applied to each colour, and can be either the source or destination α-channel (or $1 - \alpha$), zero or one. The blend operator, $\circledast$, can either be addition, subtraction, minimum or maximum of the two factors.

B.1.4 Summary

Graphics hardware is a powerful computation resource; their theoretical floating point speed make an compelling argument for adapting computation to make use of them. Some of the advantages of using graphics hardware include:

- it is highly optimised for 4-element vector arithmetic, including support for dot-products and therefore 4×4 matrix multiplication;
- parallelism through pipe-line abstraction and vertex and fragment units;
- very high bandwidth between graphics processors and graphics memory (but significantly *less* bandwidth from the system memory to graphics memory);
- support for triangle clipping to fabricate new vertices with linearly interpolated vertex attributes along clipping planes;
- a host of per-pixel operations, including depth comparison, colour manipulation, and arithmetic functions on fragment attributes.

Although much of the graphics pipe-line is programmable, it has a very specialised and unconventional programming model. For example, while fragments are free to perform arbitrary texture reads, fragment testing is not programmable and can only work with the fragment's corresponding position in the frame-buffer. Other disadvantages imposed by the graphics programming model include:

- Asymmetrical bandwidth between the graphics hardware and the processor.
- Communication between the system and graphics hardware is restricted to graphics operations, such as transferring pixels between images and describing triangle meshes.
- The hardware is free to re-sample and reinterpret pixel data, which can affect precision and perform unexpected pixel replacements with texture filters.
- References to images are always by point-sampling—never by directly addressing discrete pixels. Although this is not often a problem for images themselves,

often images can represent arrays where imprecise look-ups are undesirable. Furthermore, references to textures always succeed; texture references always return a valid texel, even if that texel is fabricated by the texture unit.

- OpenGL imposes data-flow limitations. The fragment processor is unable to write to texture memory, for example, and the fragment processor cannot reposition fragments.
- There is somewhat limited scope for fragment testing; fragments that are not removed by the fragment processor and fixed fragment-tests are always written to the frame-buffer.
- The pipe-line ordering is fixed and only subsets of the graphics pipe are programmable.

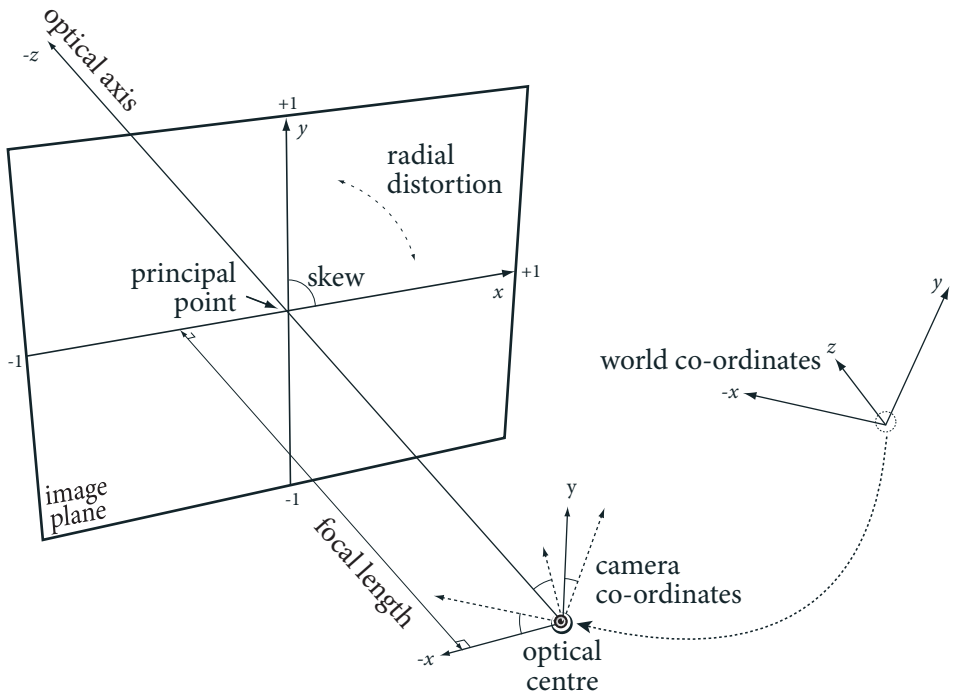


Figure B.4: The pin-hole camera model.

B.2 Camera model

The likelihoods described in Chapters 4—7 models the projection of the 3D scene onto an image plane by a pin-hole camera, illustrated in B.4. The projection $\boldsymbol{p}'$ of a 3D point $\boldsymbol{p}$ in scene-space is given by

$$\boldsymbol{p}' = \boldsymbol{P}\boldsymbol{p}, \tag{B.3}$$

where $\boldsymbol{P}$ is a 3×4 'projection matrix' [23]. The projection matrix can be decomposed into

$$\boldsymbol{P} = \boldsymbol{K}\left[\boldsymbol{R} \mid \boldsymbol{t}\right] \tag{B.4}$$

where $\boldsymbol{K}$ is the 3×3 *intrinsincs matrix*, which describes the perspective mapping induced by the camera's lens relative to the imaging plane, $\boldsymbol{R}$ is a 3×3 rotation matrix and $\boldsymbol{t}$ is a 3×1 translation vector. The intrinsics matrix is given by

$$\boldsymbol{K} = \begin{bmatrix} \alpha f & \tan(\xi) & u \\ 0 & f & v \\ 0 & 0 & 1 \end{bmatrix} \tag{B.5}$$

which describes the distance f between the imaging plane and optical centre, the principal point $[u, v]^\top$, the angle ξ between the axes of the imaging plane and the aspect ratio of the imaging frame.

Our pin-hole camera model uses the same co-ordinate frame as OpenGL. The imaging frame in this case is defined by the square $x = \pm 1$, $y = \pm 1$ in camera co-ordinates. The projection $\boldsymbol{p}' = \boldsymbol{P}\boldsymbol{p} = [x, y, w]^\top$ is within the image domain if $|\frac{x}{w}| \leq 1$ and $|\frac{y}{w}| \leq 1$. The focal length and principal point are expressed relative to this normalised image frame. Accordingly, the prior $\mathcal{U}(-0.01, 0.01)$ on the principal point in the test described in Section 4.5.3, for example, corresponds to an uncertainty of 1% in the imaging plane.

Bibliography

[1] Agrawal, M., and Davis, L. A probabilistic framework for surface reconstruction from multiple images. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), Kauai, HI, USA* (8-14 December 2001), IEEE Computer Society, pp. 470–476. ▸13, ▸28, ▸82, ▸153

[2] Baillard, C., and Zisserman, A. A plane-sweep strategy for the 3D reconstruction of buildings from multiple images. In *Proceedings of the Nineteenth International Archives of Photogrammetry and Remote Sensing (ISPRS 2000), Amsterdam, The Netherlands* (16-23 July 2000), Gopher Publishers, pp. 56–62. ▸7, ▸8

[3] Bar-Yehuda, R., and Gotsman, C. Time/space tradeoffs for polygon mesh rendering. *Association for Computing Machinery, Transaction on Graphics 15* (1996), 141–152. ▸37

[4] Blinn, J. Me and my (fake) shadow. *Computer Graphics and Applications 8* (1988), 82–86. ▸29

[5] Blinn, J. F. Models of light reflection for computer synthesized pictures. *Computer Graphics 11*, 2 (1977), 192–198. ▸138

[6] Bonet, J. S. D., and Viola, P. A. Roxels: Responsibility weighted 3D volume reconstruction. In *Proceedings of the Seventh IEEE International Conference on Computer Vision (ICCV 1999), Kerkyra, Corfu, Greece* (20-25 September 1999), IEEE Computer Society, pp. 418–425. ▸13, ▸27, ▸82, ▸153

[7] Botsch, M., Pauly, M., Kobbelt, L., Alliez, P., Levy, B., Bischoff, S., and Roessl, C. Geometric modeling based on polygonal meshes. Association for Computing Machinery Course Notes, SIGGRAPH, San Diego, August 2007. ▸37

[8] Broadhurst, A., and Cipolla, R. A statistical consistency check for the space carving algorithm. In *Proceedings of the British Machine Vision Conference (BMVC 2000)* (11-14 September 2000), British Machine Vision Association, pp. 282–291. ▸7, ▸13, ▸22, ▸153

[9] Broadhurst, A., Drummond, T., and Cipolla, R. A probabilistic framework for space carving. In *Proceedings of the Eighth International Conference on Computer Vision (ICCV 2001), Vancouver, BC, Canada* (7-14 July 2001), IEEE Computer Society, pp. 388–393. ▸27, ▸153

[10] Chernoff, H., and Lehmann, E. L. The use of maximum likelihood estimates in χ^2 tests for goodness-of-fit. *The Annals of Mathematical Statistics 25* (1954), 579–586. ▸150

[11] Collins, R. T. A space-sweep approach to true multi-image matching. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 1996), San Francisco, CA, USA* (18-20 June 1996), IEEE Computer Society, pp. 358–363. ▸14

[12] Crow, F. Shadow algorithms for computer graphics. In *Proceedings of the Fourth Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1977), San Jose, CA, USA* (20-22 July 1977), ACM Press, pp. 242–248. ▸121

[13] Culbertson, W. B., Malzbender, T., and Slabaugh, G. G. Generalized voxel colouring. In *Proceedings of the International Workshop on Vision Algorithmsm Corfu, Greece* (21-22 September 1999), vol. 1883 of *Lecture Notes in Computer Science*, Springer, pp. 100–115. ▸7, ▸18, ▸21, ▸25, ▸29, ▸153

[14] Davy, J. R., Deldari, H., and Dew, P. M. Constructive solid geometry using algorithmic skeletons. In *Proceedings of the Fifth Eurographics Workshop on Programming Paradigms in Graphics, Maastricht, The Netherlands* (2-3 September 1995), Springer, pp. 69–84. ▸158

[15] Debevec, P. E., Taylor, C. J., and Malik, J. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Computer Graphics 30*, Annual Conference Series (1996), 11–20. ▸2, ▸7, ▸8, ▸12, ▸33, ▸151, ▸155, ▸159

[16] Dick, A. R., Torr, P., and Cipolla, R. Automatic 3D modelling of architecture. In *Proceedings of the British Machine Vision Conference (BMVC 2000)* (11-14 September 2000), British Machine Vision Association, pp. 372–381. ▸3

[17] Dick, A. R., Torr, P., Ruffle, S., and Cipolla, R. Combining single view recognition and multiple view stereo for architectural scenes. In *Proceedings of the Eighth International Conference on Computer Vision (ICCV 2001), Vancouver, BC, Canada* (7-14 July 2001), IEEE Computer Society, pp. 268–274. ▸3

[18] Dyer, C. R. Image-based visualization from widely-separated views. In *Proceedings of the DARPA Image Understanding Workshop, Monterey, CA, USA* (20-23 November 1998), Morgan Kaufmann, pp. 101–105. ▸21, ▸29

[19] Eisert, P., Steinbach, E., and Girod, B. Multi-hypothesis volumetric reconstruction of 3D objects from multiple calibrated camera views. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Phoenix, AZ, USA* (15-19 March 1999), IEEE Signal Processing Society, pp. 3309–3512. ▸7, ▸19, ▸153

[20] Fischler, M. A., and Bolles, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the Association for Computing Machinery 24* (1981), 381–395. ▸2

[21] Gibson, S., Cook, J., Howard, T. L. J., and Hubbold, R. J. Icarus: Interactive reconstruction from uncalibration image sequences. In *Proceedings of the* 29th *Conference on Computer Graphics, San Antonio, Texas, 2002* (21-26 July 2002), ACM Press / ACM SIGGRAPH / Addison Wesley Longman. ▸68

[22] Habbecke, M., and Kobbelt, L. Iterative multi-view plane fitting. In *Proceedings of the Eleventh Conference on Vision, Modeling and Visualization (VMV 2006), Aachen, Germany* (22-24 November 2006), IOS Press, pp. 73–80. ▸7, ▸8

[23] Hartley, R., and Zisserman, A. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004. ▸178

[24] Heidmann, T. Real shadows real time. *Iris Universe*, 18 (1992), 28–31. ▸122

[25] Hornung, A., and Kobbelt, L. Robust and efficient photo-consistency estimation for volumetric 3D reconstruction. In *Proceedings of the Nineth European Conference on Computer Vision (ECCV 2006), Graz, Austria* (7-13 May 2006), Lecture Notes in Computer Science, Springer, pp. 179–190. ▸12, ▸22, ▸153

[26] Isidoro, J., and Sclaroff, S. Stochastic mesh-based multiview reconstruction. In *Proceedings of the First International Symposium on 3D Data Processing Visualization and Transmission* (19-21 June 2002), IEEE Computer Society, pp. 568–577. ▸7, ▸25, ▸152, ▸153

[27] Isidoro, J., and Sclaroff, S. Stochastic refinement of the visual hull to satisfy photometric and silhouette consistency constraints. In *Proceedings of the Nineth International Conference on Computer Vision (ICCV 2003), Nice, France* (14-17 October 2003), IEEE Computer Society, pp. 1335–1342. ▸7, ▸8, ▸25, ▸153, ▸154

[28] Kutulakos, K. N. Approximate n-view stereo. In *Proceedings of the Sixth European Conference on Computer Vision (ECCV 2000), Dublin, Ireland* (26 June - 1 July 2000), Lecture Notes in Computer Science, Springer, pp. 67–83. ▸7, ▸22

[29] Kutulakos, K. N., and Seitz, S. M. A theory of shape by space carving. *International Journal of Computer Vision, Marr Prize Special Issue 38*, 3 (2000), 199–218. ▸7, ▸13, ▸18, ▸20, ▸153, ▸155

[30] Laurentini, A. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence 16*, 2 (1994), 150–162. ▸9, ▸155

[31] Li, M., Magnor, M., and Seidel, H. Hardware-accelerated visual hull reconstruction and rendering. In *Proceedings of the 29th Graphics Interface Conference, Halifax, Nova Scotia* (11-13 June 2003), A K Peters, pp. 65–72. ▸11, ▸12

[32] Li, M., Magnor, M., and Seidel, H. Improved hardware-accelerated visual hull rendering. In *Proceedings of the Eigth Conference on Vision, Modeling and Visualization (VMV 2003), München, Germany* (19-21 November 2003), IOS Press, pp. 151–158. ▸12

[33] Li, M., Magnor, M., and Seidel, H. Hardware-accelerated rendering of photo hulls. *Computer Graphics Forum 23*, 3 (2004), 635–642. ▸30, ▸153

[34] Li, M., Schirmacher, H., Magnor, M., and Seidel, H. Combining stereo and visual hull information for on-line reconstruction. In *Proceedings of the IEEE Workshop on Multimedia and Signal Processing and Rendering of Dynamic Scenes, St. Thomas, VI, USA* (9-11 December 2002), IEEE Signal Processing Society, pp. 9–12. ▸13

[35] Matusik, W., Buehler, C., and McMillan, L. Polyhedral visual hulls for real-time rendering. In *Proceedings of the Twelfth Eurographics Workshop on Rendering Techniques, London, UK* (25-27 June 2001), Springer, pp. 115–126. ▸10

[36] Matusik, W., Buehler, C., Raskar, R., Gortler, S. J., and McMillan, L. Image-based visual hulls. In *Proceedings of the 27th Conference on Computer Graphics, New Orleans, Louisiana, USA* (23-28 July 2000), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 369–374. ▸11

[37] Montenegro, A. A., Carvalho, P. C. P., and Velho, L. Space carving with a hand-held camera. In *Proceedings of the Symposium on Computer Graphics and Image Processing (SIBGRAPI / SIACG 2004), Curitiba, PR, Brasil* (17-20 October 2004), IEEE Computer Society, pp. 396–403. ▸21, ▸153

[38] Neal, R. M. Probabilistic inference using markov chain monte carlo methods. Tech. Rep. CRG-TR-93-1, University of Toronto, 1993. ▸35, ▸66

[39] O'Loughlin, J., and O'Sullivan, C. Real-time animation of objects modelled using constructive solid geometry. In *Proceedings of the Fifteenth Spring Conference on Computer Graphics (SCCV 1999), Budmerice, Slovakia* (28 April - 1 May 1999), Comenius University, pp. 67–73. ▸121, ▸158

[40] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., and Purcell, T. J. A survey of general-purpose computation on graphics hardware. In *Proceedings of the European Conference on Computer Graphics, Dublin, Ireland* (29 August - 2 September 2005), Eurographics Association, pp. 21–51. ▸4, ▸37, ▸46, ▸171

[41] Phong, B. T. Illumination for computer generated pictures. *Comm. ACM 18*, 6 (June 1975), 311–317. ▸162

[42] Piegl, L., and Tiller, W. *The* nurbs *Book*. Springer-Verlag, 1995. ▸156

[43] Ponce, J., Chelberg, D. M., and Mann, W. B. Analytical properties of generalized cylinders and their projections. In *Proceedings of the DARPA Image Understanding Workshop, Los Angeles, CA, USA* (February 1987), Morgan Kaufmann, pp. 340–350. ▸156

[44] Porter, T., and Duff, T. Compositing digital images. In *Proceedings of the Eleventh Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1984), July, 1984* (July 1984), ACM Press, pp. 253–259. ▸27

[45] Potmesil, M. Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing 40*, 1 (1987), 1–29. ▸10

[46] Prock, A. C., and Dyer, C. R. Towards real-time voxel coloring. In *Proceedings of the DARPA Image Understanding Workshop, Monterey, CA, USA* (20-23 November 1998), Morgan Kaufmann, pp. 315–321. ▸21, ▸29, ▸153

[47] Sainz, M., Bagherzadeh, N., and Susin, A. Carving 3D models from uncalibrated views. In *Proceedings of the Fifth IASTED International Conference on Computer Graphics and Imaging, Kauai, HI, USA* (12-14 August 2002), IASTED/ACTA Press, pp. 144–149. ▸153

[48] Sainz, M., Bagherzadeh, N., and Susin, A. Hardware accelerated voxel carving. In *Proceedings of the First Ibero-American Symposium in Computer Graphics (SIACG 2002), Guimarães, Portugal* (1-5 July 2002), Eurographics Portuguese Chapter, pp. 289–297. ▸29, ▸153

[49] Saito, H., and Kanade, T. Shape reconstruction in projective grid space from large number of images. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 1999), Ft. Collins, CO, USA* (23-25 June 1999), IEEE Computer Society, pp. 49–54. ▸20

[50] Segal, M., and Akeley, K. The OpenGL graphics system: A specification. Tech. rep., Silicon Graphics Computer Systems, 1992. ▸47, ▸171

[51] Seitz, S., and Dyer, C. Photorealistic scene reconstruction by voxel coloring. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 1997), San Juan, Puerto Rico* (17-19 June 1997), IEEE Computer Society, pp. 1067–1073. ▸7, ▸13, ▸14, ▸15, ▸20, ▸153

[52] Seitz, S., and Dyer, C. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision 25*, 3 (November 1999), 151–173. ▸153

[53] Sivia, D. S. *Data Analysis: A Bayesian Tutorial.* Oxford University Press, 1996. ▸34

[54] Slabaugh, G., Culbertson, B., Malzbender, T., and Schafer, R. Improved voxel colouring via volumetric optimization. Tech. Rep. 3, Center for Signal and Image Processing, Georgia Institute of Technology, 2000. ▸7, ▸13, ▸24, ▸153

[55] Slabaugh, G., Malzbender, T., and Culbertson, W. Volumetric warping for voxel colouring on an infinite domain. In *Proceedings of the Second European Workshop on 3D Structure from Multiple Images of Large Scale Enviroments, Dublin Ireland* (1-2 July 2000), Lecture Notes on Computer Science, Springer, pp. 109–123. ▸20, ▸153

[56] Smelyansky, V. N., Morris, R. D., Kuehnel, F. O., Maluf, D. A., and Cheeseman, P. Dramatic improvements to feature-based stereo. In *Proceedings of the Seventh European Conference on Computer Vision (ECCV 2002), Copenhagen, Denmark* (28-31 May 2002), Lecture Notes in Computer Science, Springer, pp. 247–261. ▸7, ▸29, ▸148, ▸153

[57] Smith, A. R. A pixel is not a little square, a pixel is not a little square, a pixel is not a little square! (and a voxel is not a little cube!). Tech. Rep. 6, Microsoft, 1995. ▸23

[58] Snow, D., Viola, P., and Zabih, R. Exact voxel occupancy with graph cuts. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2000), Hilton Head, SC, USA* (13-15 June 2000), IEEE Computer Society, pp. 345–353. ▸11

[59] Steinbach, E., Eisert, P., Girod, B., and Betz, A. 3D Object reconstruction using spatially extended voxels and multi-hypothesis voxel coloring. In *Proceedings of the Fifteenth International Conference on Pattern Recognition (ICPR 2000), Barcelona, Spain* (3-8 September 2000), IEEE Computer Society, pp. 774–777. ▸22

[60] Steinbach, E., Girod, B., Eisert, P., and Betz, A. 3D reconstruction of real-world objects using extended voxels. In *Proceedings of the International Conference on Image Processing (ICIP 2000), Vancouver, BC, Canada* (10-13 September 2000), IEEE Computer Society, pp. 569–572. ▸22

[61] Stevens, M. R., Culbertson, W. B., and Malzbender, T. A histogram-based colour consistency test for voxel colouring. In *Proceedings of the Sixteenth International Conference on Pattern Recognition (ICPR 2002)* (11-15 August 2002), IEEE Computer Society, Quebec City, Canada, pp. 118–121. ▸7, ▸13, ▸22, ▸24, ▸153

[62] Szeliski, R. Rapid octree construction from image sequences. *Computer Vision and Image Understanding 58*, 1 (July 1993), 23–32. ▸10, ▸155

[63] Thormählen, T., Broszio, H., and Mikulastik, P. Robust linear auto-calibration of a moving camera from image sequences. In *Proceedings of the Seventh Asian Conference on Computer Vision, Hyderabad, India* (13-16 January 2006), Lecture Notes in Computer Science, Springer, pp. 71–80. ▸140, ▸142

[64] Thormählen, T., Broszio, H., and Weissenfeld, A. Keyframe selection for camera motion and structure estimation from multiple views. In *Proceedings of the Eighth European Conference on Computer Vision (ECCV 2004), Prague, Czechoslovakia* (11-14 May 2004), Lecture Notes in Computer Science, Springer, pp. 523–535. ▸140, ▸142

[65] Tran, S., and Davis, L. 3D surface reconstruction using graph cuts with surface constraints. In *Proceedings of the Nineth European Conference on Computer Vision (ECCV 2006), Graz, Austria* (7-13 May 2006), Lecture Notes in Computer Science, Springer, pp. 219–231. ▸13, ▸26, ▸153

[66] Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. Bundle adjustment – A modern synthesis. In *Vision Algorithms: Theory and Practice*, W. Triggs, A. Zisserman, and R. Szeliski, Eds., Lecture Notes in Computer Science. Springer Verlag, 2000, pp. 298–375. ▸2

[67] Tsai, R. Y. An efficient and accurate camera calibration technique for 3D machine vision. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 1986), Miami, FL, USA* (22-26 June 1986), IEEE Computer Society, pp. 364–374. ▸73, ▸135

[68] Vogiatzis, G., Torr, P., and Cippola, R. Multi-view stereo via volumetric graph-cuts. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), San Diego, CA, USA* (June 20-26 2005), IEEE Computer Society, pp. 391–399. ▸7, ▸8, ▸26, ▸153, ▸154

[69] William, S. A., Press, H., Flannery, B. P., and Vetterling, W. T. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1993. ▸146

[70] Yang, R., and Pollefeys, M. Multi-resolution real-time stereo on commodity graphics hardware. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003), Madison, WI, USA* (16-22 June 2003), IEEE Computer Society, pp. 211–217. ▸30, ▸153

[71] Yang, R., Pollefeys, M., Yang, H., and Welch, G. A unified approach to real-time multi-resolution. *International Journal of Image and Graphics 6* (2004), 565–575. ▸29

[72] Yang, R., Welch, G., and Bishop, G. Real-time consensus-based scene reconstruction using commodity graphics hardware. In *Proceedings of the Tenth Pacific Conference on Computer Graphics and Applications, Beijing, China* (9-11 October 2002), ICCC Computer Society. ▸29, ▸153