Efficient Sampling of Textured Scenes in Vertex Tracing

Thomas Ullmann

IC:IDO VR productive, Germany
Thomas.Ullmann@icido.de

Thomas Preidel, Beat Bruderlin

Technical University of Ilmenau, Dept. of Computer Science and Automation
Thomas.Preidel/Beat.Bruederling@prakinf.tu-ilmenau.de

Abstract

We present vertex tracing, an adaptive progressive ray tracing approach for efficient sampling of the radiance function, based on refinement in object space and subsequent reconstruction, using standard 3D graphics accelerator hardware.
The main focus of this paper is the reconstruction of reflected and transmitted texture maps. By taking advantage of the newest graphics hardware features (such as the pixel shader) even higher levels of recursion can be supported without explicit sampling of the textures. In addition, the graphics hardware is used for an efficient visibility test, which leads to a further reduction of ray samples, and for the rendering of diffuse local illumination effects, as well as for progressive rendering of the global illumination effects, which are superimposed over the hardware-rendered scene in object space. With this approach, interactive performance for realistically rendered illumination effects of scenes of average complexity, can be achieved on a standard PC with off-the-shelf 3D graphics accelerators.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Vertex Tracing, Adaptive Progressive Ray Tracing, Radiance Reconstruction, Texturing

1. Introduction

Most PCs purchased today come with a 3D graphics hardware accelerator as a standard option. Such graphics hardware is based on raster conversion of polygons, supporting visibility tests, Gouraud shading and texture processing, as basic functionality. Real-time animation of dynamic scenes of average complexity (consisting of up to several hundred thousand to a few million polygons) is achieved. The level of realism of hardware rendering, however, is limited to simple local illumination effects. Indirect reflections, refractions and certain other global illumination phenomena are not directly supported. Moreover, scenes with increasingly high numbers of polygons can no longer be visualized in real-time with hardware-based on raster conversion.

Ray tracing, on the other hand, can achieve much higher levels of realism. While it is still far from being able to reach real-time performance levels on a single-processor PC, ray tracing can be easily parallelized, taking advantage of cache coherence and thus outperforming raster conversion hardware for highly complex scenes with several million polygons [18, 8]. Therefore ray tracing is becoming a viable alternative to raster-based graphics hardware for visualization of large scenes.

Adaptive ray tracing can significantly reduce the number of samples that need to be calculated, and further improve performance. One problem with adaptive ray tracing, however, is the integration of textures. Textures are most often used to approximate the material properties of surfaces that result in a high frequency radiance function. However, by reducing the useable image coherence in adaptive sampling of such functions, the computation cost rises significantly. How to circumvent the direct sampling of primary and secondary reflections and refractions is one of the main topics of this

paper. Only the exploitation of modern graphics hardware, in particular the programmable pixel shading functionality, permits an efficient integration of textures in the vertex tracing approach.

After a short survey of related work on the topic of adaptive progressive ray tracing in section 2, section 3 gives a brief summary of vertex tracing and introduces an efficient visibility test. Section 4 describes the details of the hardware-supported reconstruction of textured scenes. An evaluation of the results is given in section 5, and the conclusion and suggestions for further research in section 6.

2. Related work

One of the first adaptive progressive ray tracers was proposed by PAINTER and SLOAN[9], and provided for handling the image as a continuous region without pixel borders. The authors aimed for the reconstruction of a fast low-quality image, as well as a high-quality *anti-aliased* one through optimal sample distribution in the image space. For this purpose, they applied sampling, based on refinement of the image with a two-dimensional BSP tree, which was refined until the desired level of quality was achieved. The samples generated were interpolated in the image space with*Delaunay triangulation*[3]; no special handling of textures took place.

In [5], GUO presented a modified method of adaptive progressive sampling. GUO used "*directional coherence maps*" (DCM) for efficient handling of radiance discontinuities. With DCMs, image discontinuities are refined by the divide and conquer principle until only simple directional discontinuity edges can be approximately represented and subsequently interpolated with *oriented finite elements* . In [14] the approach from [5] was extended with respect to more efficient texture handling. The inclusion of texture values doesn't occur until the interpolation phase, after sampling has been completed. However, the representation of secondary (reflected or transmitted) textures is not possible in this approach.

While the methods in [1, 2, 5, 6, 11, 14, 9] rely on sampling in the image space, the method of [15] operates in object space. TELLER *et al.* use *interpolants*, which are constructed about the scene objects and represent their radiance. It is possible to further refine the interpolants as needed, in order to better approximate both discontinuities and non-linearities. Typical for this object space-based method is the possibility of exploiting both object space coherence and temporal coherence. The disadvantage of the method is that the construction of interpolants can only be done for convex scene objects. Furthermore, textures are not specially taken into account, so that complete sampling of textures occurs.

Vertex tracing, first proposed in [16], likewise belongs to the group of samplers that pursue object space-based refinement. The data structure of the refinement is directly oriented toward the geometry of *arbitrarily tessellated* objects. This fact also has a positive effect on the handling of textures. Furthermore, in contrast to [14], it can even entirely do without sampling of secondary textures. Through the use of graphics hardware in the form of a pixel shader, an efficient hybrid form of raster graphics and ray tracing results.

3. Vertex Tracing

The fundamental principle of the vertex tracing technique [17] is shown in Fig. 1. Similar to numerous methods for accelerating ray tracing, vertex tracing also exploits coherence characteristics, in this case primarily image coherence. The core

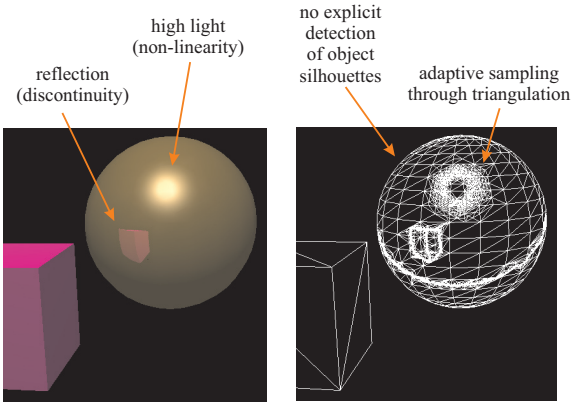


Figure 1: *Adaptive progressive refinement in object space - principle of the vertex tracing*

approach.

of the approach is a targeted sampling of the radiance function in order to achieve a significant reduction in the number of ray intersection tests necessary for generating the image. The sampling operates in an adaptive, progressive manner.

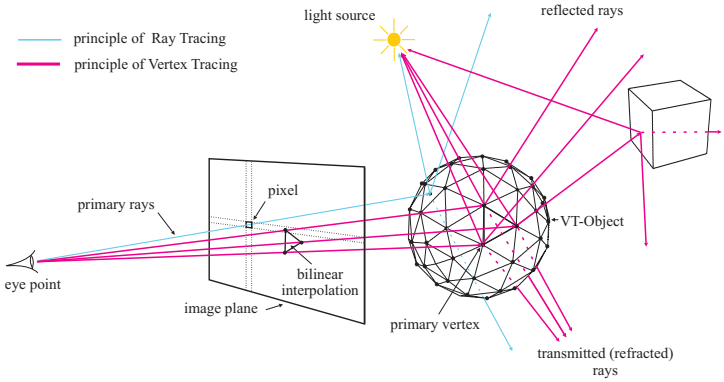


Figure 2: *Concept of vertex tracing. In contrast to traditional ray tracing all primary rays are generated directly from the object vertices (and from the adaptively refined object geometry, respectively).*

This means that nonlinearities or discontinuities contained in the radiance function are approximated more exactly in each step. Using the example of the sphere in Fig. 1, an adaptive sampling of the radiance values can be observed at those positions where a high variance in the radiance function occurs (reflection, refraction, shadow edges).

In contrast to [10, 3], the generation of the support geometry for the sampling for vertex tracing takes place in object space, with subsequent projection in the image space. Additionally, in contrast to [15], the support geometry is directly superimposed on the existing geometry of the triangulated scene objects, which undergo an additional adaptive triangulation. As shown in Fig. 2 (red ray path), the primary rays are directly assigned to the object vertices, from which further rays are sent out. The primary vertices of the scene objects thus form the initial sample pattern necessary for the sampling.

The analysis of the results of the ray tracing (of the samples) influences the refinement of the object geometry. Secondary vertices are generated, which in turn serve as the starting point for a new sampling. The resulting sample values are finally interpolated bi-linearly via graphics hardware and progressively displayed for each refinement step.

A detailed description of the vertex tracing method can be found in [16, 17]. To summarize, the method features the following advantages or disadvantages, respectively:

- A collision test of primary rays is obsolete, since the geometry of the VT objects, and hence their support points (*vertices*), serves as the basis for further ray tracing.
- No additional object detection is necessary, since the primary object geometry is used as the starting point for further triangulation.
- A combination of hardware-based rendering and ray tracing is possible, since the representation of objects generated via ray tracing likewise is done by hardware-supported rendering.
- Even in the case of few available samples, the rendered scene yields a relatively good qualitative impression, since the objects are maintained in their original geometry and can be directly displayed as a *"quick preview"*.
- Sampling of primary or secondary textures can be avoided through refinement in object space and the use of hardware rendering.
- In comparison to ray tracing (which behaves linearly), the run-time behavior of vertex tracing is sub-linear with respect to the number of pixels of the image.
- Vertex tracing scales linearly with the scene complexity, while ray tracing has a sub-linear time complexity.

3.1. Hardware Accelerated Visibility Test

Because the primary rays are directly assigned to the primary vertices, no visibility test needs to be carried out for primary rays, in contrast to standard ray tracing. The visibility test must therefore take place separately before the assignment of primary rays, in order to avoid unnecessary sampling of hidden primary vertices. After every change in camera position, the visibility test is carried out as a pre-process before the actual vertex tracing. It significantly determines the density of the initial sample pattern.

The test uses a modified form of the ID buffer principle [20], as shown schematically in Fig. 3. Each face f_i of a VT (vertex tracing) object is assigned an unambiguous color ID and

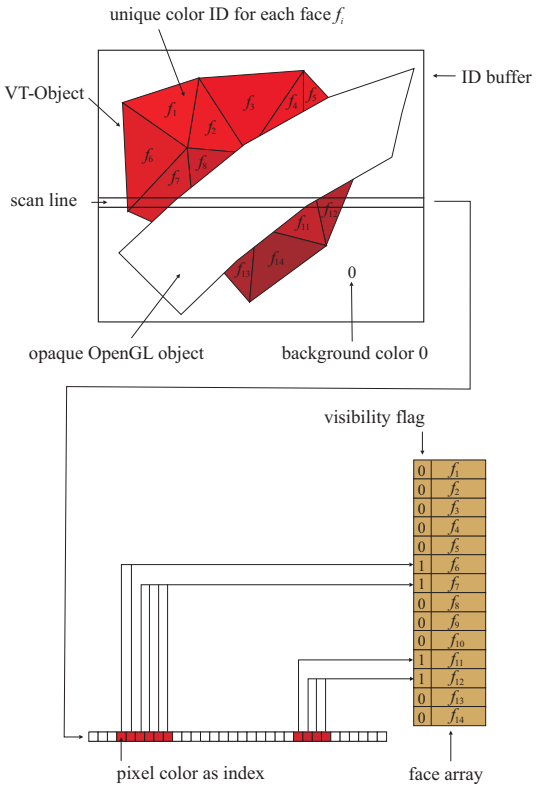


Figure 3: *ID buffer test. The color ID of a pixel is used as an index for the face array. Visibility flags are set only for indexed faces.*

rendered into the ID buffer. Objects that are not VT objects (all other OpenGL objects) are assigned the color ID 0, as is the background of the buffer. After the ID buffer has been read into the main memory, each pixel acts as an index for indexing a contiguous array of faces of all VT objects with visibility flag set to 0. Scan line by scan line and pixel by pixel, the ID buffer is then processed and the visibility flag of the face to which the index points set to 1. All faces visible by at least one pixel are then marked as *visible*. From the visibility of each face f_i, conclusions about the corresponding vertices can then be drawn.

4. Reconstruction Without Explicit Texture Sampling

To reconstruct the sampled radiance function, hardware-based Gouraud shading is used, carrying out linear interpolation between the sample values. Similar to numerous ray tracers we use for shading a slightly modified Phong shading model [22] to compute the local component I_{local} on each vertex $\vec{v}_i$ (see Fig. 4). If we add the two global components I_{refl} and I_{trans}, the overall intensity $I(\vec{v}_i)$ of a vertex $\vec{v}_i$ becomes:

$$I(\vec{v}_i) = I_{local}(\vec{v}_i) + k_r I_{refl}(\vec{p}_r) + k_t I_{trans}(\vec{p}_t). \quad (1)$$

$I_{refl}(\vec{p}_r)$ represents the light intensity that arrives due to reflection from a point $\vec{p}_r$ in $\vec{v}_i$, taking into consideration the attenuation factor k_r. The same applies for the intensity $I_{trans}(\vec{p}_t)$ and the attenuation factor k_t for transmission. Be-

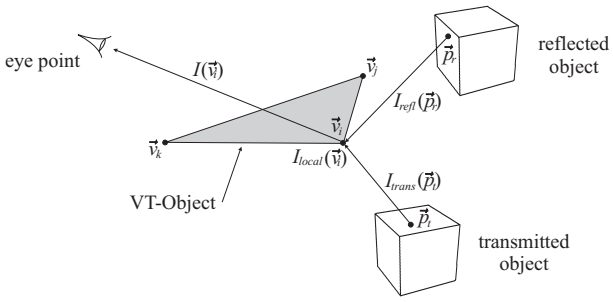


Figure 4: *Principle of shading in vertex tracing.*

cause the ray tracing is carried out recursively, $I(\vec{v}_i)$ must also be recursively determined. Taking Eqn. 1 as a basis, this results in the following recursion for the example of the reflection components $I(\vec{p}_r)$:

$$I(\vec{p}_r^{\,j}) = I_{local}(\vec{p}_r^{\,j}) + k_r^{j+1} I_{refl}(\vec{p}_r^{\,j+1}) + k_t^{j+1} I_{trans}(\vec{p}_t^{\,j+1}), \quad j = 0, 1, ..., n \quad (2)$$

where j is the number of recursion steps.

To reconstruct the sampled radiance function, hardware-based Gouraud shading is used, carrying out linear interpolation between the sample values. The faces generated from refinement are passed to the graphics hardware and displayed accordingly. Vertex tracing is characterized by its hybrid rendering character. Both rendering on the basis of local illumination models in the form of OpenGL rendering hardware, and on the basis of a global illumination model in the form of ray tracing, can be realized in parallel. The core issue in this context is the differentiation of the scene objects. Specularly reflecting objects are defined as VT objects and subjected to vertex tracing. Objects that reflect primarily diffusely remain "normal" scene objects (non-VT objects)[†] and are rendered with conventional OpenGL rendering.

Textures, which are highly effective in the approximation of complex material characteristics in traditional hardware-based rendering or ray tracing, actually turn out to be rather detrimental in adaptive ray tracing. The sometimes high-frequency texture information may lead to increased sample

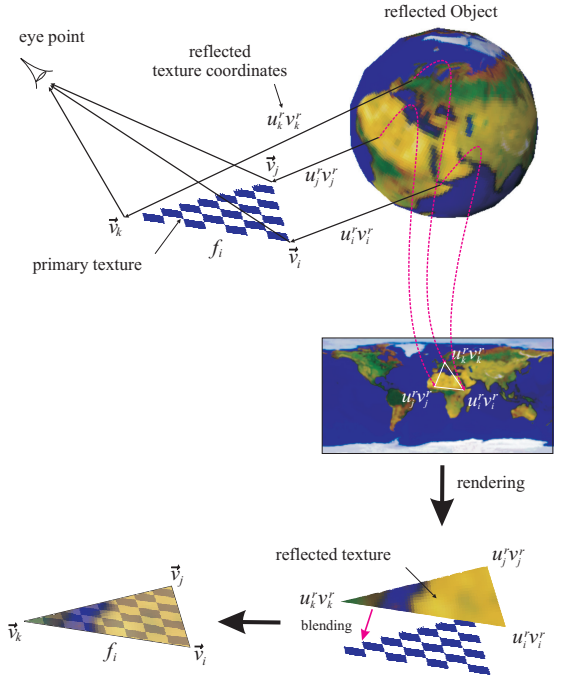


Figure 5: *Integration of textures in vertex tracing. We get a* defined face, *if all vertices point to the same face with one texture.*

generation, since the usable image coherence is significantly reduced. A highly varying radiance function can result from the continuous radiance distribution of an object representation after texturing. The advantage of adaptive sampling, i.e. the reduction of the number of samples, could be dramatically weakened. Methods such as proposed in [19] attempt to deal with this problem, by carrying out measurements in texture space, from which further criteria for effective sampling are derived.

A by far more efficient method than [19] is the additional exploitation of object space information. If, for each sample, only the corresponding texture coordinates are calculated, foregoing an immediate texture lookup, adaptive refinement

[†] Diffuse objects may also be defined as VT objects, if other global illumination phenomena (e.g. shadows) are to be simulated on these objects.

can also be carried out without consideration of the radiance values saved in the texture. In our approach, the actual texture lookup occurs after refinement via hardware-based rendering, in that an interpolation of the texture coordinates between the samples takes place.

Fig. 5 shows the fundamental approach with respect to the use of textures in vertex tracing. Depending on the refinement process, new faces f_i are generated. The control of the refinement occurs without involving radiance values from textures. In the example of Fig. 5, for each vertex $\vec{v}_i$ of face f_i, only the corresponding texture coordinates u_i^r, v_i^r (if they exist) of the reflecting objects are determined and stored accordingly. The texture lookup does not occur until the actual rendering process. Each face f_i is then rendered with its primary texture into the frame buffer. If a reflected texture also exists, a rendering of f_i with this texture additionally takes place. A suitable blend operation finally mixes the textures in the frame buffer.

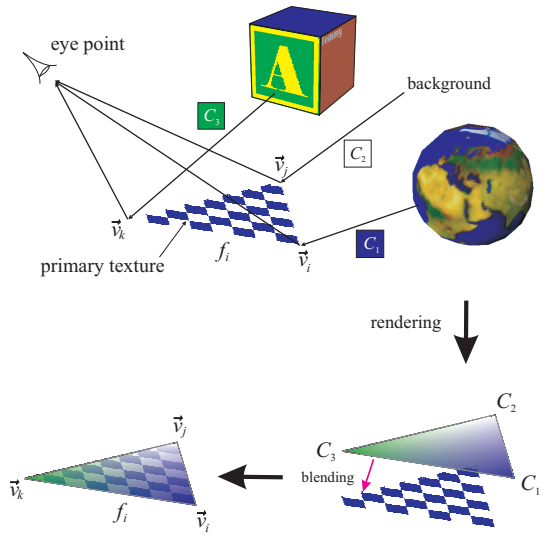


Figure 6: *In case of different textures reflected from one face we perform a texture lookup and interpolate the colors (case of* undefined faces*).*

The assignment of texture coordinates for a transmitted object occurs analogous to the process for a reflected one. If there are both reflected and transmitted textures, up to three textures, including the primary texture, must be blended over another in the frame buffer. With this method, sampling of textures is completely avoided in the first recursion step. This approach would also be conceivable in each further step, but depends strongly on the existing hardware prerequisites, since more than three textures per face would have to be rendered. This is the reason for complete sampling of textures based on their radiance values in recursion steps $>$ 1.

If every vertex of a face f_i features the same reflected or transmitted texture, this is called a "*defined face*." If texture coordinates from different textures of different objects appear, or if certain vertices have even no texture coordinates, "*undefined faces*" result. As shown in Fig. 6, in this case the radiance values are read directly from the texture. According to Eqn. 1, the determined color value, in combination with the object's primary color, determines the intensity at the vertex $\vec{v}_i$.

4.1. Multi-Pass Texturing

Once the texture coordinates of reflected and transmitted objects have been determined at each vertex of a defined face f_i, it is rendered. Based on Eqn. 1, the final fragment color C_{frag} in the frame buffer is determined by:

$$C_{frag} = C_{prim}CT_{prim} + \alpha_r(C_{refl}CT_{refl} + C_{r,comb}) + \alpha_t(C_{trans}CT_{trans} + C_{t,comb}) \quad (3)$$

with $C_{prim}, C_{refl}, C_{trans}$ the object's primary color, CT_{prim}, CT_{refl}, CT_{trans} the texture color, and α_r, α_t the alpha value as attenuation factor of the reflection or transmission, respectively. It is assumed that the object color from the modulation (GL_MODULATE) results from the primary color and the texture color. C_{comb} represents the combined color starting with the second recursion step, resulting from an additional reflection or transmission.

With simultaneous rendering of up to three textures on a face, the application of multi-texturing is preferred. One problem, however, is the widespread hardwiring of texture units in graphics hardware. In OpenGL before Version 1.2, a freely programmable combination of texture units was largely impossible. The executable fragment operations are pre-defined and do not allow mapping of Eqn. 3 (see also [21]).

Without more flexible programming of the texture units, as already supported in current OpenGL extensions and applied in Section 4.2, the only alternative remains *multi-pass* texturing. In this case, rendering according to Eqn. 3 would have to take place in up to five passes. In order to nonetheless avoid such enormously high rendering costs, Eqn. 3 can be slightly modified. Already combining $C_{r,comb}$ and C_{refl}, or $C_{t,comb}$ and C_{trans}, before the texture modulation results in

$$C_{frag} = C_{prim}CT_{prim} + \alpha_r((C_{refl} + C_{r,comb}) \cdot CT_{refl}) + \alpha_t((C_{trans} + C_{t,comb}) \cdot CT_{trans}). \quad (4)$$

Certainly, Eqn. 4 represents only an approximation of 3. The modified color combination nonetheless allows addition before the actual rendering process, on the basis of which the multi-pass texturing can be reduced to three passes. For each pass, the primary object color is modulated with the texture color in a texture unit, and written to the frame buffer. Starting with the second cycle, writing takes into consideration α-*blending*, as already shown in Fig. 5. On the one hand,

the blend operation acts as an attenuation factor α_r or α_t , respectively; on the other, as an addition according to Eqn. 4.

4.2. Multi-Texturing

Only on modern graphics subsystems with integrated *pixel shader* has it become possible, to carry out true multi-texturing, as required by vertex tracing, in a single pass. The representation of the texture shading function according to Eqn. 4, however, cannot itself be explicitly carried out with the functional scope of a pixel shader. The crux of the problem lies in the number of available color input registers. For one pixel shader operation, only two colors can be defined, the *primary color* and the *secondary color*; furthermore, the latter must do without an alpha value. In the case of four texture units and two color inputs, for example, this means that one color component is still missing for representation of the shading function.

This can be remedied with the aid of the *pass-through*[4]. According to Eqn. 4, three texture units are used by CT_{refl}, CT_{trans}, and CT_{prim}. If the graphics subsystem possesses a minimum of four texture units, the third color component from 4 can be represented with the aid of pass-through.

If the graphics subsystem possesses six or more texture units, representation of Eqn. 3 is, on the basis of the pass-through function, conceivable in a single pass. However, if fewer than four texture units are available, at least two rendering passes must be carried out. The addition between the individual passes can again take place with blend operations.

5. Results

Vertex tracing was shown to significantly reduce the number of samples through the use of graphics hardware. The introduction of hardware-supported texturing allows a further reduction in samples in the context of adaptive sampling. By exploiting object space information in the form of texture coordinates and using the methods presented here, textures can even be included in higher recursion steps without explicit sampling of their radiance function. Fig. 7 shows an example of this. Despite the radiance variance that appears in the chessboard texture (discontinuities at the borders between light and dark fields), no denser sampling is necessary for the texture reflections of the cup. The area of the textures mirroring each other in 7b has been only minimally sampled.

With the presented method of texture integration we achieve computation rates in real-time or close to real-time of textured scenes. The cup shown in figure 7a with an image resolution of 1280x1024 on a Dual-Athlon 1.35 GHz system is rendered at preview quality in 130 ms (with or without textures); for a full quality image the rendering takes about 1.5 s. The total number of samples determined is about 47’000. In contrast, it takes about 12.7 s to ray trace the same image with a conventional ray tracing approach, using the same data structures and intersection algorithm.

A combination of reflecting and transmitting textures is shown in Fig. 8a. The simultaneous representation of a primary texture of the cup would likewise be conceivable here. Fig. 8b further demonstrates the method’s efficiency. By sampling only the primary vertices (preview representation), the texture of the wall (the rendering of which occurs with standard OpenGL) on the table surface, or that of the chessboard pattern in the cup, is almost completely represented as a reflection, so that a high degree of image quality is again achieved at interactive frame rates.

A detrimental effect, which appears due to rasterization, can be seen in Fig. 9. Specifically, Fig. 9a shows a partially incorrect perspective representation of the reflected table texture, in contrast to the correct representation via ray tracing in 9b. The greatest distortion occurs especially in the lower area of the cup, where the texture is stretched the most. This effect appears more strongly when a reflecting or transmitting texture is represented with only a few samples. The cause of the distortions is the incorrect perspective correction by the hardware in the case of reflecting or transmitting textures. One solution would be denser sampling in the critical areas, as proposed in [7]. Methods able to do without a higher number of samples will be investigated in further work.

The proposed visibility test enables discarding all non-visible vertices and thus avoids unnecessary sampling of hidden scene geometry. The run time of the visibility test depends strongly on the size of the image to be rendered, in addition to the scene complexity. Because the frame buffer is read from the graphics hardware, the bottleneck is located here. Even with modern hardware, the buffer read time is approximately 20-40 ms at a resolution of 1024x768 pixels. The run time of the actual scan line methods from Fig. 3 is relatively insignificant in comparison (less than 10 ms at 1024x768 resolution for the examples shown). Of greater influence, however, is the rendering time necessary for the visibility test (first rendering pass), which in turn depends on the complexity of the scene. Without the implementation of culling mechanisms, rendering time increases linearly with the size of the scene. Significant here is that with the aid of the alpha channel, the entire rendering process can be reduced to a *single* pass. For this purpose, the scene is immediately rendered into the frame buffer in "normal" fashion for the visibility test, thus simultaneously serving as the final representation. The primary faces are separately marked with a code in the alpha channel. The scan line of the frame buffer then relatively reliably filters out all primary faces and sets the corresponding visibility flags.

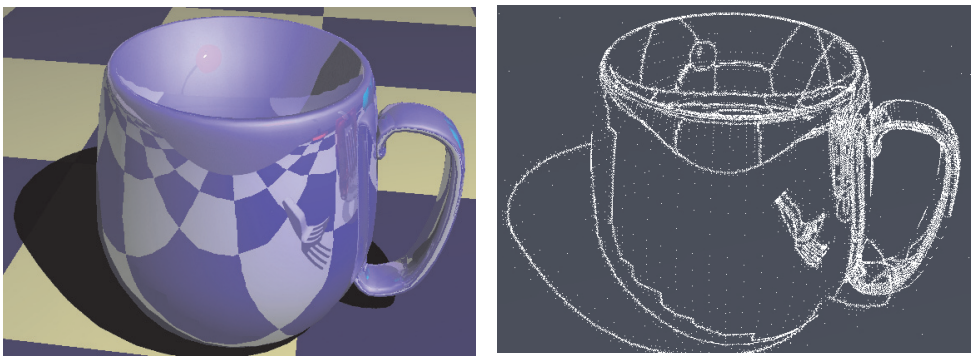
6. Conclusions

Vertex tracing allows not only the combination of OpenGL rendering and ray tracing, but additionally offers an efficient integration of Textures, through refinement in object space. Despite the generally high radiance variance in textures, it requires no additional sampling, even for textures from higher recursion steps. The representation of the shading function in a single texture rendering pass (multi-texturing) was made possible by exploiting modern rendering hardware in the form of the pixel shader. Additionally, vertex tracing's run-time performance was improved by the introduction of the visibility test. The test allows a determination of only relevant samples to avoid a sampling of invisible features.

Due to its sub-linear performance with respect to pixels, vertex tracing is suitable for supplementing global illumination effects in traditional hardware-based rendering. Its efficient sampling and the possibility of selecting individual VT objects are of benefit especially in virtual reality applications featuring geometrically accurate reflection investigations, for example.

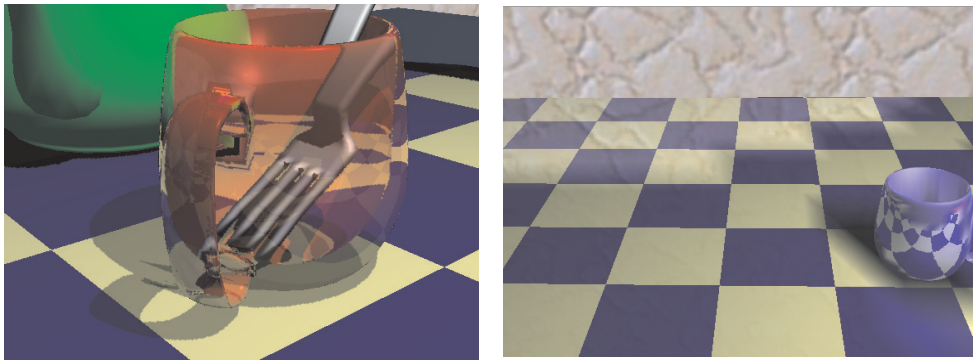
References

1. John Amanatides and Alain Fournier, *Ray casting using divide and conquer in screen space*, International Conference on Engineering and Computer Graphics, August 1984, pages 290–296. f21g (1984). 2
2. T. Akimoto, K. Mase, and Y. Suengaga, *Pixel-selected ray tracing*, IEEE Computer Graphics (1991), 11:14–22. 2
3. Y. Eldar, M. Lindenbaum, M. Porat, and Y. Zeevi, *The farthest-point strategy for progressive image sampling*, In Proceddings, 12th International Conference on Pattern Recognition, Jerusalem (1994). 2, 3
4. Sebastien Domine and John Spitzer, *Texture shaders*, Tech. report, nVidia Corporation, http://developer.nvidia.com, 2000. 6
5. Baining Guo, *Progressive radiance evaluation using directional coherence maps*, Computer Graphics (SIGGRAPH 1998 Proceedings) (1998), 255–266. 2
6. Frederik W. Jansen and Jarke J. van Wijk, *Fast previewing techniques in raster graphics*, Proceedings Eurographics (1983), 195–202. 2
7. Manuel M. Oliveira, *Correcting Texture Mapping Errors Introduced by Graphics Hardware*, Proceedings Pacific Graphics '01 (2001), 31–38. 6
8. M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan, *Rendering complex scenes with memory-coherent ray tracing*, In SIGGRAPH 97 Conference Proceedings, Annual Conference Series, pages 101–108 (1997). 1
9. James Painter and Kenneth Sloan, *Antialiased ray tracing by adaptive progressive refinement*, Computer Graphics (SIGGRAPH '89 Proceedings) (1989), 281–288. 2
10. F. Pighin, Dani Lischinski, and David Salesin, *Progressive previewing of ray-traced images using image-plane discontinuity meshing*, Rendering Techniques (1997), 115–126. 3
11. Gunther Raidl and Wilhelm Barth, *Fast adaptive previewing by ray tracing*, In Proceedings of 12th Spring Conference on Computer Graphics (1996). 2
12. John Spitzer, *Programmabel texture blending*, Tech. report, nVidia Corporation, http://developer.nvidia.com, 2000.
13. John Spitzer, *Register combiners*, Tech. report, nVidia Corporation, http://developer.nvidia.com, 2000.
14. Annette Scheel, Marc Stamminger, Jörg Pütz, and Hans-Peter Seidel, *Enhancements to directional coherence maps*, url: citeseer.nj.nec.com/492134.html. 2
15. Seth Teller, Kavita Bala, and Julie Dorsey, *Conservative radiance interpolants for ray tracing*, Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering) (1996). 2, 3
16. Thomas Ullmann, Daniel Beier, Alexander Schmidt, and Beat Brüderlin, *Adaptive progressive vertex tracing in distributed environments*, In Proceedings of Pacific Graphics '01, Tokyo, Japan (2001). 2, 3
17. Thomas Ullmann, Alexander Schmidt, Daniel Beier, and Beat Brüderlin, *Adaptive progressive vertex tracing for interactive reflections*, In Proceedings of EuroGraphics '01, Short Presentations, Manchester, UK (2001). 2, 3
18. I. Wald, C. Benthin, M. Wagner, and P. Slusallek, *Interactive rendering with coherent ray tracing*, Proceedings of EuroGraphics 2001 (2001). 1
19. Theo van Walsum, Peter R. van Nieuwenhuizen, and Frederik W. Jansen, *Refinement criteria for adaptive stochastic ray tracing of textures*, In Proceedings of Eurographics '91, page 155-166, Amsterdam (1991). 4
20. H. Weghorst and G. Greenberg, *Improved computational methods for ray tracing*, ACM Transactions on Graphics (1984), 52–69. 3
21. Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner, *Opengl programming guide, third edition*, Addison Wesley, 1999. 5
22. Alan Watt and Mark Watt, *Advanced animation and rendering techniques, theory and practice*, Addison-Wesley Publishing Company, Inc., 1992. 4



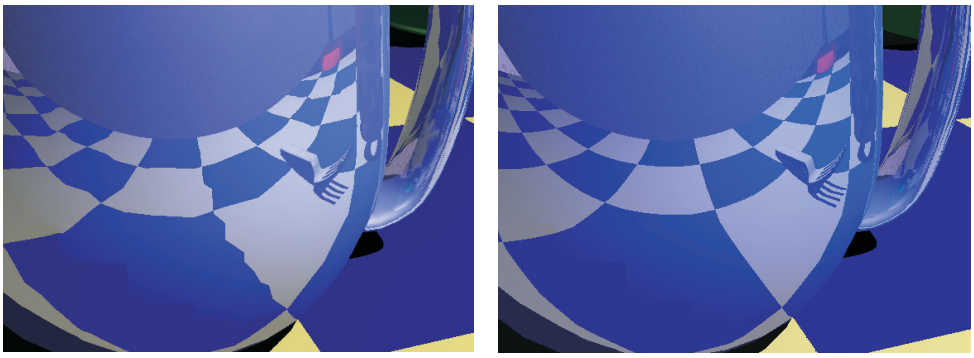
a) b)

Figure 7: *No sampling is performed for the radiance variance of textures. (Here, the reflection of the checkerboard in the cup doesn't need any additional samples.)*



a) b)

Figure 8: *a) Combination of transmitted and reflected textures. b) In case that only the primary vertices were sampled an almost complete representation of the reflected wall texture can be shown (preview representation shown)*



a) b)

Figure 9: *a) Incorrect perspective representation of reflected or transmitted textures in Vertex Tracing. b) Correct representation in case of standard Ray Tracing.*