Real-Time Volumetric Tests Using Layered Depth Images

Matthias Trapp† & Jürgen Döllner‡

Hasso-Plattner-Institute, University of Potsdam, Germany

Abstract

This paper presents a new approach for performing efficiently 3D point-in-volume tests for solid and arbitrary complex shapes. It classifies a 3D point as inside or outside of a solid specified by 3D polygonal geometry. Our technique implements a basic functionality that offers a wide range of applications such as clipping, collision detection, interactive rendering of multiple 3D lenses as well as rendering using multiple styles. It is based on an extension of layered depth images (LDI) in combination with shader programs. An LDI contains layers of unique depth complexity and is represented by a 3D volume texture. The test algorithm transforms a 3D point into an LDI texture space and, then, performs ray marching through the depth layers to determine its classification. We show how to apply real-time volumetric tests to implement 3D clipping and rendering using multiple styles. In addition, we discuss limitations and possible improvements.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Graphics processors; I.3.5 [Computer Graphics]: Boundary representations; I.3.6 [Computer Graphics]: Graphics data structures and data types;

1 Introduction

There is a need for basic 3D geometric operations that can be performed in real-time for input data that shows high geometric complexity or is large in size. A point-in-volume test is such kind of operation that offers a broad range of applications in computer graphics and visualization. For example, for rendering 3D volumetric lenses [VCWP96] it is necessary to decide which fragment or vertex is inside the lens volume. Current rendering techniques [RH04] enable such functionality using expensive image-based multipass rendering algorithms executed per rendering frame. Further, despite image-based *Constructive Solid Geometry* (CSG) [KD05], there is currently no approach that performs fast pixel-precise clipping against multiple arbitrary shaped meshes.

To enable such general functionality within a single rendering pass, we present a volumetric test algorithm that targets real-time rendering applications in particular. This test is easy to implement and can be used in all programmable pipeline stages.

For this purpose, we research an adequate data structure for the volumetric representation of static, solid meshes with arbitrary shapes that facilitates an efficient volumetric test. Since this representation can be created in pre-processing, no additional rendering pass at runtime is necessary.

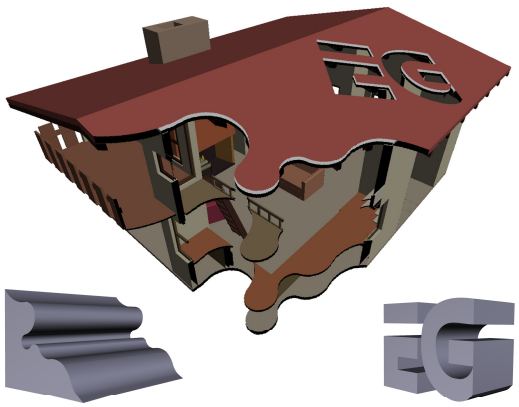

Figure 1: *An example for using volumetric tests to perform pixel-precise clipping against two complex 3D shapes in a single rendering pass.*

† matthias.trapp@hpi.uni-potsdam.de

‡ doellner@hpi.uni-potsdam.de

This paper is structured as follows. Section 2 gives an overview of the related work. Section 3 presents the method of our volumetric test. Section 4 sketches the implementation and Section 5 discusses the results. In Section 6 we give conclusions and ideas for future work.

2 Related Work

The idea of LDIs is presented in [SGwHS98]. An LDI is a view of the scene from a single input camera view but with multiple pixels along each line of sight. The size of the representation grows only linearly with the observed depth complexity in the scene.
The depth peeling algorithm for order-independent transparency was introduced by [Cas01]. It uses a second depth test to extract layers of unique depth complexity from an arbitrary scene. It is possible to re-use this layers by performing a render-to-volume technique [Dro07]. In [Lef03] various memory layout options and optimizations are discussed. In this context, ray marching is a well known algorithm for interactive volume rendering [ZRL*07].
A dynamic approach for slice-based object voxelization is presented in [Eis06]. It can be applied in a single rendering pass but lacks accuracy.

3 Concept

To perform the proposed test in real-time, our approach consists of two components: a data structure that is fully accelerated by graphics hardware and an algorithm that operates on instances of these. The algorithm is implemented in a shader program. The components are used in the following manner:

1. Creation of an image-based representation of the shapes volume that stores only its depth values along a viewing ray that is aligned towards the negative z-axis. We denote this representation as *Volumetric Depth Sprite* (VDS). This step is performed during pre-processing and its result is stored by using high precision textures (see Section 3.1).
2. At shader runtime, we perform a *Volumetric Parity Test* (VPT) for each point. Therefore, a point is transformed into the specific VDS coordinate system and, then, is tested against all depth values stored in the particular VDS (see Section 3.2).

This approach requires solid shapes and, since this step is performed in pre-processing, we claim the shapes representation to be a static, non-animated mesh. Due to the recently established *Unified Shader Model*, which uses a consistent instruction set across all shader types, it became possible to apply the VPT in all programmable pipeline stages.

3.1 Volumetric Depth Sprite

A VDS extends the concept of LDIs [SGwHS98]. LDIs can be represented on graphics hardware via a 3D texture or a 2D texture array [Bly06]. For our application, they contain layers of unique depth complexity. Figure 2 shows an example

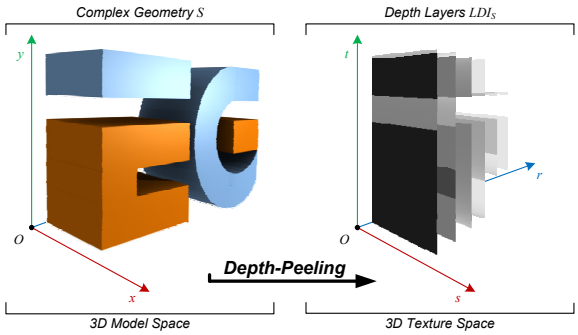

Figure 2: *An example of a complex geometric shape and the generated depth layers of the associated volumetric depth sprite. The shape has a maximal depth complexity of 6.*

of a VDS derived from a complex 3D shape. A VDS representation of a shape S consists of the following components:

$$VDS_S = (P_S, \mathbf{M}_S, LDI_S, d_S)$$

Where $P_S \in \mathbb{R}^3$ denotes the position of the sprite in world coordinates. The matrix $\mathbf{M}_S$ represents linear transformations such as rotation and scaling of the VDS. The depth complexity of S is denoted as $d_S \in \mathbb{N}_{/\{0,1\}}$.
To obtain a depth value $d_i \in [0;1] \subset \mathbb{R}, 0 \leq i \leq d-1$ in the i^{th}-depth layer for the 2D point (s,t), we sample the 3D texture in LDI texture space with the coordinate $(s,t,(1/d)\cdot i)$.

3.2 Volumetric Parity Test

Given a VDS, the VPT classifies a point $P \in \mathbb{R}^3$ with respect to its position in relation to the shape's volume. It can be either inside or outside the volume. For reasons of precision, we do not consider the case that the coordinate can be on the border of the shape.
Before testing the parity of P, it must be transformed into the specific LDI texture space of the VDS. For example, if P is a fragment in clip space, this transformation can be obtained by:

$$T = (T_s, T_t, T_r) = \mathbf{M}_S \cdot \left(\left(\mathbf{O}^{-1} \cdot P \right) - P_S \right)$$

Where $\mathbf{O}$ denotes the current orientation matrix. At first, P is transformed into world coordinates, then, translated into the LDI texture space and finally adjusted by the VDS model matrix. The volumetric test can be formulated as follows:

$$VPT(T, VDS_S) = p_T = \begin{cases} 1, & \exists d_i \in LDI_S : d_i \leq T_r \leq d_{i+1} \\ 0, & otherwise \end{cases}$$

To model the above test, we introduce a Boolean *coordinate parity* $p_T \in \{0,1\}$. Starting with a initial parity, the ray $R = [(T_s, T_t, 0)(T_s, T_t, 1)]$ marches through the depth layers of the LDI, compares T_r with the stored depth values d_i, and swaps p_T every time it crosses a layer of unique depth complexity (see Figure 4).

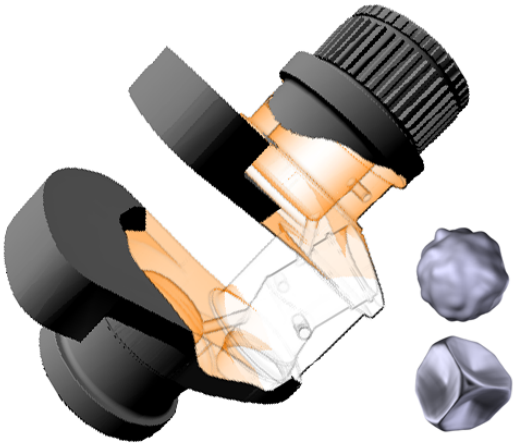

Figure 3: *Possible application of the volumetric test. Blueprint rendering is combined with X-Ray shading and Gouraud shading.*

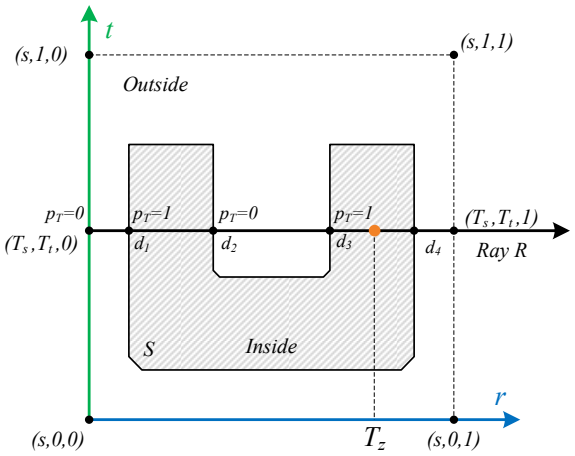

Figure 4: *Ray marching in 3D LDI texture space seen from the s-axis. The parity of the coordinate T swaps if the ray R passes a depth layer. S has a depth complexity of 4.*

4 Implementation

Our implementation is based on OpenGL [Mar06] in combination with GLSL [Kes06]. For *render-to-texture* (RTT), we use framebuffer objects, high precision 32bit float textures, and floating point depth buffer precision.

4.1 Depth Peeling to 3D Texture

The creation of a VDS is performed within a pre-processing step using multipass RTT. Given a shape S, we generate the associated LDI_S by performing the following steps:

1. Transformation of the shape into the unit volume $[0;1]^3$. An orthogonal projection is set that covers this unit volume. The near and far clipping planes are adjusted accordingly.
2. Determine depth complexity d_S and create a 3D texture with a certain width w, height h, and depth d_S. To enable precise sampling results, we setup nearest neighbor interpolation. The border of the the texture is set 1 for each color component and the texture wrap mode is set to *clamp to border*. To keep the texture size of the 3D texture within bounds, our implementation uses a luminance texture format with a single 32bit floating point channel.
3. Depth-peel [Cas01] the solid S using RTT for each slice of a 3D texture separately. It is necessary to linearize the depth values using inverse W-buffer [LJ99].

4.2 Ray-Marching Shader Implementation

Our implementation of the ray-marching algorithm needs to iterate over the number of texture slices in the 3D texture. Therefore, it demands a shader model 3.0 compliant graphics hardware. Figure 5 shows the GLSL source code that implements the VPT.

5 Results

Our technique has been implemented based on the Virtual Rendering System VRS, which was used to create the presented figures. Despite collision detection, our approach has a number of applications in real-time rendering. Figure 1 shows pixel-precise clipping against multiple complex volumes performed in a single rendering pass. Figure 3 demonstrates the binding of different visual appearances to different VDS. It uses a combination of blueprint-rendering [ND04], X-Ray shading, and Gouraud shading.

5.1 Performance

Our test platform is a GeForce 8800 GTS with 640 MB video memory. We are able to render the depicted scenes at interactive frame rates. Table 1 displays the frames per seconds (FPS) for each scene. The achieved frame rates depend

```
bool volumetricParityTest(
  in vec4      T,          // coordinate in LDI-space
  in sampler3D LDI,        // layered depth image LDIs
  in int       depth,      // depth complexity ds
  in bool      initParity) // initial parity p
{
  // initial parity; true = outside
  bool  parity = initParity;
  // calculate offset to address texture slices
  float offset = 1.0 / float(depth);
  // for each texture layer do
  for(float i = 0.0; i < float(depth); i++)
  { // perform depth test
    if(T.r < texture3D(LDI, vec3(T.st, offset * i)).x)
    {
      parity = !parity; // swap parity
    }
  }
  return parity;
}
```

Figure 5: *GLSL implementation of the volumetric parity test. It exploits dynamic flow control features.*

Table 1: *Comparative performance evaluation of the volumetric test for different scenes and volumetric depth sprites.*

Model	#Samples	w/o Test	with Test
House	10	74.4	**73.2**
Crank	80	8.6	**7.5**

on the number of VDS used, thus the number of samples (#Samples) the VPT has to perform. The presented volume test consists of less than 20 assembler instructions per executed loop. The time consumption for pre-processing a shape depends on its depth complexity and the desired texture dimension. For our test cases the VDS creation takes up to two seconds per shape.

5.2 Limitations

The key drawback of our method is the high space complexity of an LDI. However, tests for different resolutions of the 3D texture exposed that a texture size of 1024^2 pixels deliver results of fairly high quality. Larger texture resolution does not improve the quality significantly. When using complex non-convex solids, the VPT might fail for too low resolutions (128^2 pixels). The presented figures use an LDI resolution of 1024^2 pixels.
Further, undersampling can be a problem for planes nearly parallel to the LDI direction. This produces artifacts such as fuzzy edges. To compensate this drawback one could apply multisampling that increases the number of samples at the same time.
Performing depth peeling with high texture resolution for a number of complex shapes cannot be done in real-time. Due to this, our approach is limited to static meshes because animated shapes require a re-computation of its VDS.

6 Conclusions & Future Work

We presented an approach for performing fast point-in-volume tests for arbitrary 3D solids in real-time. It employs an image-based representation of shape's volume as well as a parity test that can be implemented on modern GPU efficiently. We demonstrated the flexibility of our approach in different application examples.
The key problem concerns the precision of the LDI. The accuracy of the volume test depends on the used texture resolution and depth precision. Increasing the texture resolution leads to a high spatial complexity of the VDS. Therefore, it is necessary to research a lossless compression scheme for 3D textures containing depth information.
Further, it is possible to perform an optimal viewpoint selection for VDS creation to determine the viewing parameters that will result in a minimal depth complexity. Additionally, it is desirable to obtain not only a boolean value but also the distances of the point with respect to the shape border in relation to the unit volume. This can be achieved by adding an additional pre-processing step that encodes this information into each texel in the VDS.

Acknowledgments

This work has been funded by the German Federal Ministry of Education and Research (BMBF) as part of the InnoProfile research group "3D Geoinformation"' (www.3dgi.de).

References

[Bly06] BLYTHE D.: The Direct3D 10 System. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), ACM Press, pp. 724–734.

[Cas01] CASS EVERITT: *Interactive Order-Independent Transparency*. Tech. rep., NVIDIA Corporation, 2001.

[Dro07] DRONE S.: Real-Time Particle Systems On the GPU in Dynamic Environments. In *SIGGRAPH '07* (New York, NY, USA, 2007), ACM, pp. 80–96.

[Eis06] EISEMANN, ELMAR AND DÉCORET, XAVIER: Fast Scene Voxelization and Applications. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2006), ACM SIGGRAPH, pp. 71–78.

[KD05] KIRSCH F., DÖLLNER J.: OpenCSG: A Library for Image-Based CSG Rendering. In *Proceedings of USENIX 2005* (2005), pp. 129–140.

[Kes06] KESSENICH J.: *The OpenGL Shading Language Language Version: 1.20 Document Revision: 8*, September 2006.

[Lef03] LEFOHN A.: Interactive Visualization of Volumetric Data on Consumer PC Hardware. In *Tutorial, IEEE Visualization* (2003).

[LJ99] LAPIDOUS E., JIAO G.: Optimal Depth Buffer for Low-Cost Graphics Hardware. In *HWWS '99* (New York, NY, USA, 1999), ACM, pp. 67–73.

[Mar06] MARK J. KILGARD: *NVIDIA OpenGL Extension Specifications*. Tech. rep., NVIDIA, November 2006.

[ND04] NIENHAUS M., DÖLLNER J.: Blueprints: Illustrating Architecture and Technical Parts Using Hardware-Accelerated Non-Photorealistic Rendering. In *GI '2004* (2004), pp. 49–56.

[RH04] ROPINSKI T., HINRICHS K.: Real-Time Rendering of 3D Magic Lenses Having Arbitrary Convex Shapes. In *WSCG* (February 2004), vol. 12, pp. 379–386.

[SGwHS98] SHADE J., GORTLER S., WEI HE L., SZELISKI R.: Layered Depth Images. In *SIGGRAPH '98* (New York, NY, USA, 1998), ACM, pp. 231–242.

[VCWP96] VIEGA J., CONWAY M. J., WILLIAMS G., PAUSCH R.: 3D Magic Lenses. In *UIST '96* (New York, NY, USA, 1996), ACM Press, pp. 51–58.

[ZRL*07] ZHOU K., REN Z., LIN S., BAO H., GUO B., SHUM H.-Y.: *Real-Time Smoke Rendering Using Compensated Ray Marching*. Tech. Rep. MSR-TR-2007-142, Microsoft Research, September 2007.