

Semiautomatic Shader Code Generation for Rendering Voxelized Polygonal Models

Jan Fischer*
University of Victoria, Canada

David Whittaker†
University of Victoria, Canada

Aaron Lefohn
Intel Corp., Seattle, USA

Bruce Gooch
University of Victoria, Canada

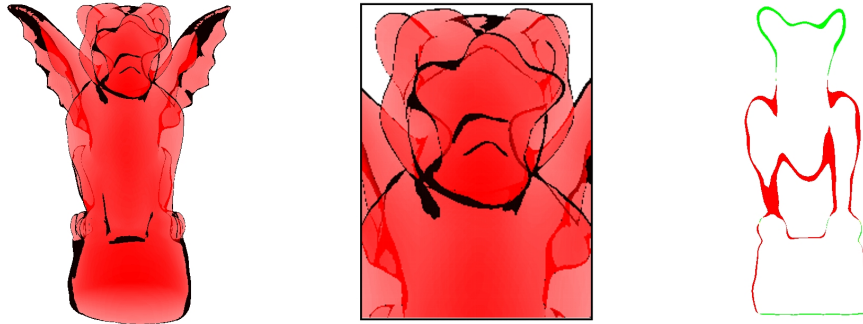


Figure 1: Gargoyle model voxelized and rendered with our system. A voxelization with a total resolution of 352 slices was generated in a single pass. The right image shows one of the slices, with front-facing voxels shown red, and back-facing voxels shown green.

Abstract

Voxelization is an important basic operation for many applications in computer graphics and related areas. We describe a voxelization method for general, complex models, which generates a high-resolution volume in a single pass. One interesting application is the simultaneous display of all layers of a model, e.g., in illustrative visualization. Due to the representation of the voxelization data as bitstrings encoded in multiple textures, writing rendering code based on it can be a tedious task. We therefore, present an automated code generation technique that abstracts and encapsulates binary data access. This allows us to quickly write programs that generate renderings based on voxel data; we call these “voxel shaders”.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms

Keywords: Voxelization, shader code generation

1 Description

In certain computer graphics applications, it makes sense to explicitly generate and process surface pixels for all parts of a model, including those that are hidden from direct view. While previous work on *voxelization* focused on applications such as shadow rendering or refraction [Eisemann and Décoret 2006], we see illustrative visualization as an important motivation for extracting all surface layers of an object. In many types of illustrative rendering, the display of inner structures and hidden layers is desired. This poster deals with the real time generation and rendering of high-

resolution voxelizations. Our voxelization process is executed entirely on the GPU and only requires a short amount of time. This makes it possible to re-voxelize the observed object or scene in every frame. Therefore, volumetric representations can be generated in each frame for animated and deforming models. Our implementation is capable of generating a volume with 352 slices in a single pass. An example voxelization generated by our system is shown in Fig. 1.

Due to the encoding of the voxel data as binary information in the individual color components of several textures, rather lengthy shader code has to be written to access it. We describe an automated code generation technique for rendering shaders that access voxelization data. The code generation is based on a minimal extension of the shading language with a small number of voxelization-specific keywords. This makes it easier to write associated shaders, and we show results from more complex shaders that were written using our language extensions. We call rendering shaders accessing voxel data using our language extensions “voxel shaders”. The keywords, which we have chosen to have an XML-like look, are replaced with the corresponding full shader code. The following code fragment demonstrates the core section of our illustrative visualization shader. Here, the additional keywords have the typical tag format `<tag>|</tag>`:

```
<VOLUME>
  <SEGMENTCOUNT>8</SEGMENTCOUNT>
  <BITSPERSEGMENT>22</BITSPERSEGMENT>
  <VOLUMECOUNT>2</VOLUMECOUNT>

  numFrontFaces += V1FRONT_SLICE(i);
  innerSlices += step(0.1, numFrontFaces);
  numFrontFaces -= V1BACK_SLICE(i);
  if(edgeAtten == 0 && V2FRONT_SLICE(i)) {
    edgeAtten = innerSlices;
  }
</VOLUME>
```

This work was supported by a Postdoctoral Research Fellowship from the German Research Foundation (DFG).

References

EISEMANN, E., AND DÉCORET, X. 2006. Fast Scene Voxelization and Applications. In *Proc. of ACM Symposium on Interactive 3D Graphics*, 71–78.

*e-mail: jan@janfischer.com

†e-mail: greystone@gmail.com