June 01, 2004

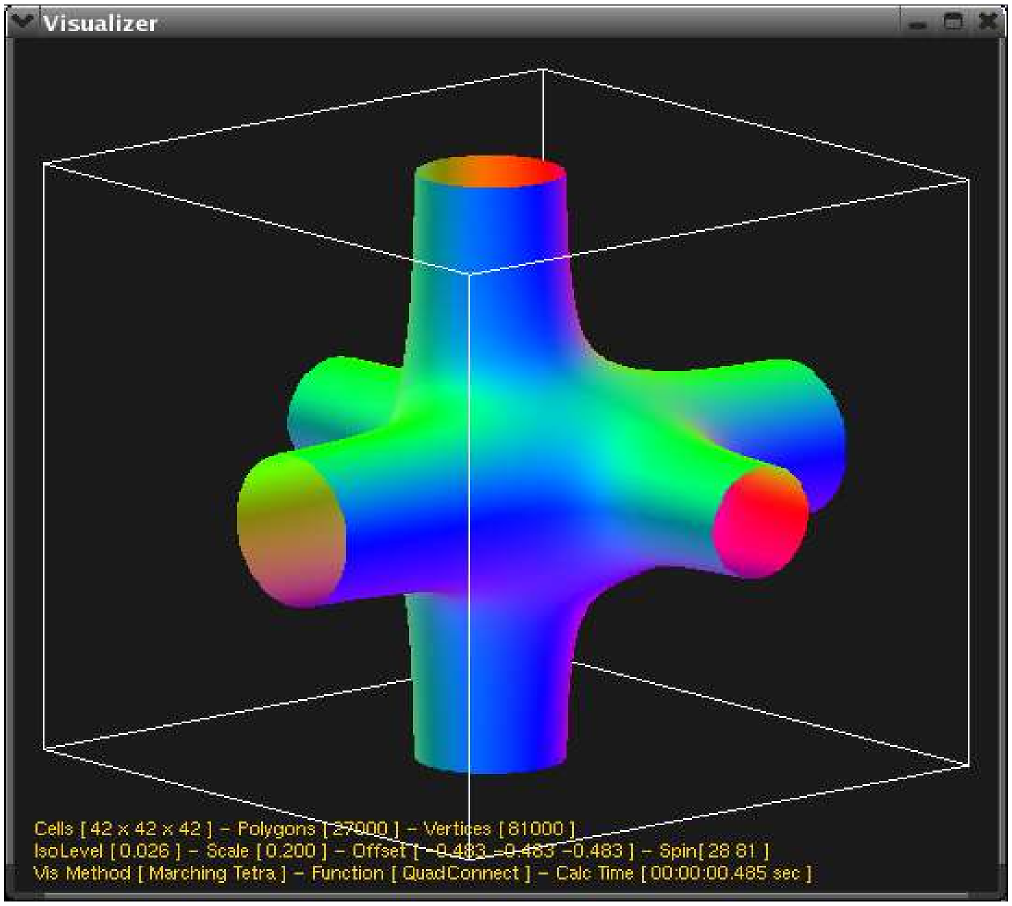
Visualizer

An implicit surface rendering application

Derek Gerstmann - C1405511
MSc Computer Animation
NCCA – Bournemouth University

OVERVIEW OF APPLICATION

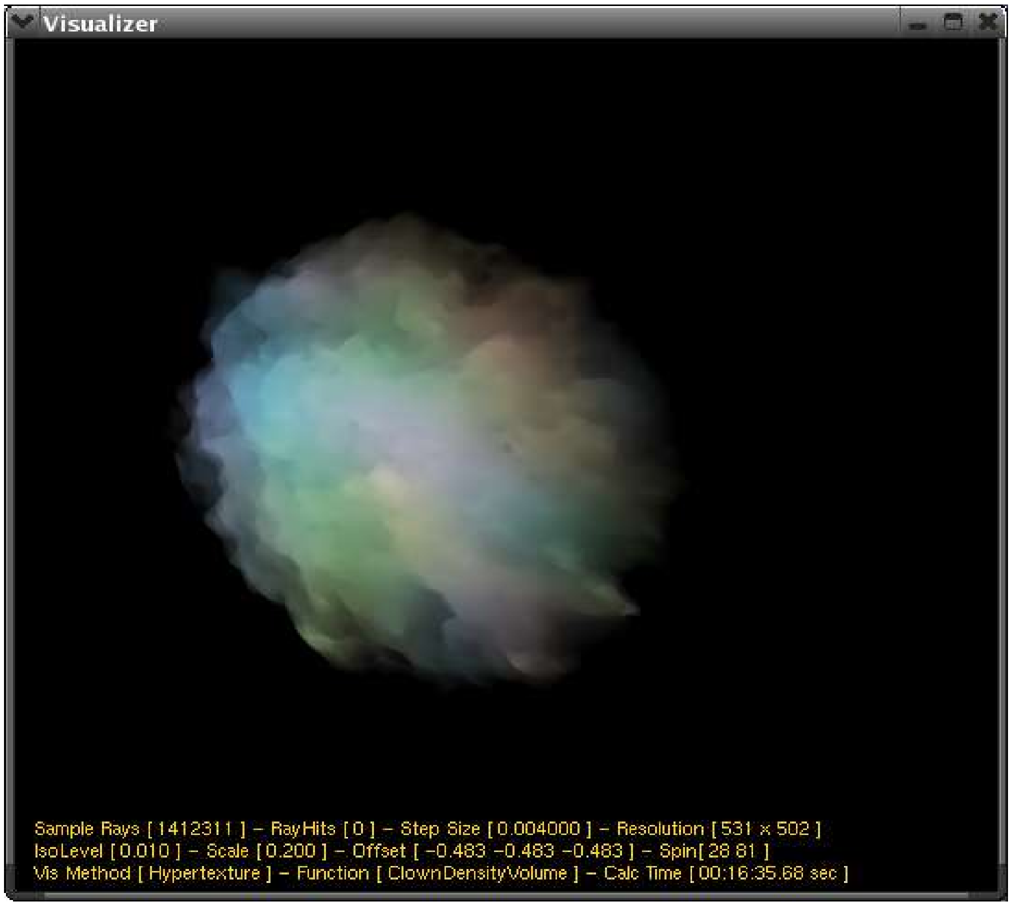
Visualizer is an interactive application capable of rendering implicit models, including both surfaces and volumes. There are two polygonizing methods (marching cubes, and marching tetrahedrons) which can be used to construct a geometric representation, in addition to a scanline raytracer, and a renderer for hypertextures.



```
Cells [ 42 x 42 x 42 ] - Polygons [ 27000 ] - Vertices [ 81000 ]
IsoLevel [ 0.026 ] - Scale [ 0.200 ] - Offset [ -0.483 -0.483 -0.483 ] - Spin[ 28 81 ]
Vis Method [ Marching Tetra ] - Function [ QuadConnect ] - Calc Time [ 00:00:00.485 sec ]
```

FIGURE 1 Visualizer running in marching tetrahedral polygonizatoin mode (DG).

The following sections detail the various aspects of the application, including insights into it's design and the development, as well as the fundamentals behind the algorithms used.



```
Sample Rays [ 1412311 ] – RayHits [ 0 ] – Step Size [ 0.004000 ] – Resolution [ 531 x 502 ]
IsoLevel [ 0.010 ] – Scale [ 0.200 ] – Offset [ -0.483 -0.483 -0.483 ] – Spin[ 28 81 ]
Vis Method [ Hypertexture ] – Function [ ClownDensityVolume ] – Calc Time [ 00:16:35.68 sec ]
```

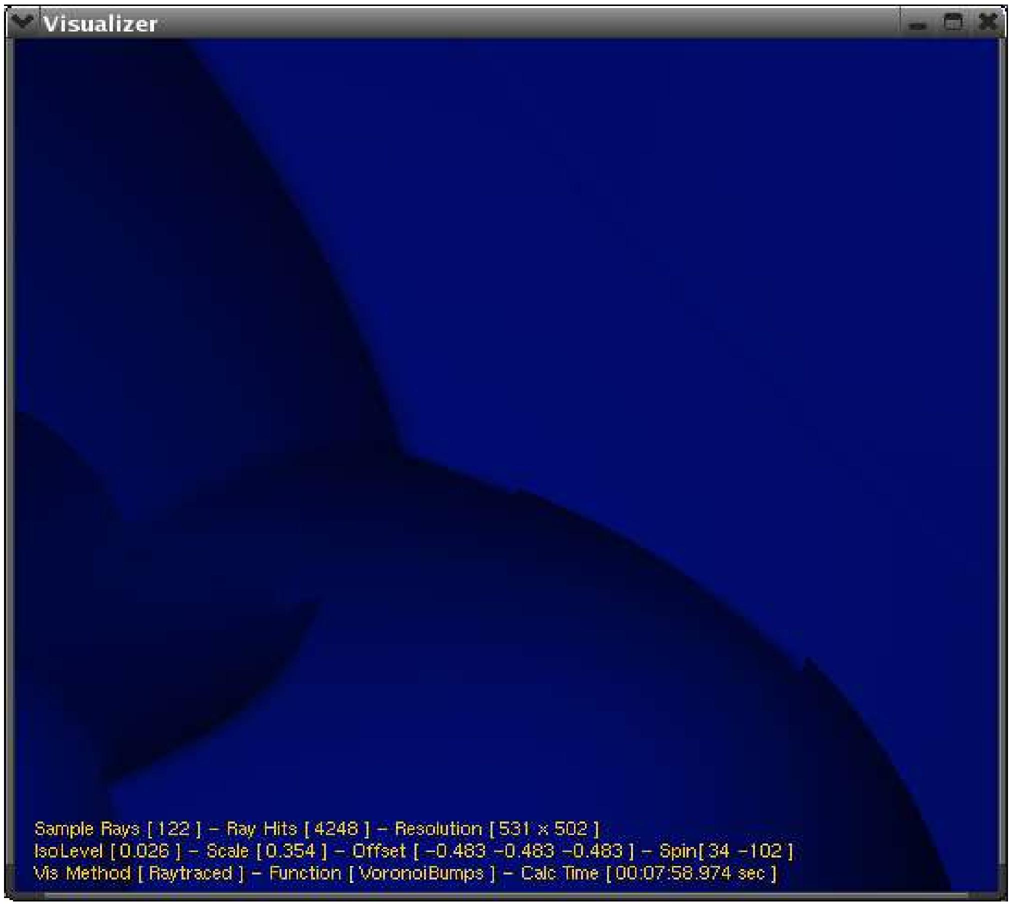
FIGURE 2 Visualizer running in hypertexture visualization mode (DG).

FEATURES AND USAGE

Visualizer was designed as a simple graphical application, and uses a single window as the basis for it's interaction.

After loading, the application selects a random implicit function and enters into the marching tetrahedron visualization mode. At this point, the application generates a geometric model, and shades it with standard specular shading.

The default state is animated, and the geometric model will be continually re-polygonized at new positions in space. This mode can be toggled by hitting the 'm' key.



```
Sample Rays [ 122 ] – Ray Hits [ 4248 ] – Resolution [ 531 x 502 ]
IsoLevel [ 0.026 ] – Scale [ 0.354 ] – Offset [ -0.483 -0.483 -0.483 ] – Spin[ 34 -102 ]
Vis Method [ Raytraced ] – Function [ VoronoiBumps ] – Calc Time [ 00:07:58.974 sec ]
```

FIGURE 3 Visualizer running in ray traced visualization mode (errors due to undersampling!) (DG).

The view within the main window can also be modified, allowing the user to rotate about the object and reveal a different section by rotating the object about it's primary axes. This is enabled by clicking in the main window and dragging the mouse.

In additon, the default orthographic project can be switched to a fixed perspective view with a 45 degree fov by hitting the 'p' key. In this mode, the camera can be moved closer or farther away from the discretized volume by hitting the 'w' and 's' keys, respectively.

In addition to the mouse movements, a set of keyboard shortcuts provide the user with access to other commands. These are outline in the table below.

Keyboard Shortcut	Command
'i'	**Toggle info display on/off (HUD)**
'b'	**Toggle Bounding Box on/off**
'j'	**Toggle Hypertexture spherical fadeout.** *This has the effect of diminishing the density of the volume as it approaches a fixed radius, thus fading it out.*
'k'	**Toggle Hypertexture keep in view.** *This will restrict the evaluation of the hypertexture to a spherical region located directly in front of the camera.*
'+/-'	**Increase/Decrease the resolution.** *The effect of the resolution varies depending on the visualization. For polygonizers, it increases the cell count for discretization, for ray tracing, it increases the sample rays, and for hypertexture, it descreases the step size for ray marching.*
'e'	**Toggle Hypertextue visualization mode.**
'r'	**Toggle RayTraced visualization mode.**
't'	**Toggle Marching Tetrahedron polygonization and visualization mode.**
'y'	**Toggle Marching Cubes polygonization and visualization mode.**
'[' ']'	**Scale the visualization up/down.**
'n'	**Toggle color by normals, or standard opengl shading.**
'l'	**Toggle wireframe rendering mode.**
'f'	**Select a new random implicit function.** *Note that the implicit functions for polygonization and raytracing differ from the density functions for hypertexture.*
'w'	Increase the camera's position in Z.
's'	Decrease the camera's position in Z.
'm'	Toggle animation.

Keyboard Shortcut	Command
PAGE UP/PAGE DOWN	**Increase/Decrease IsoSurface level.** *Only applicable for polygonization.*
HOME/END	Increase/Decrease Function Offset in Z.
UP/DOWN	Increase/Decrease Function Offset in Y.
LEFT/RIGHT	Increase/Decrease Function Offset in X.

FEATURES AND USAGE

To maintain platform independence, the application was designed modularly and with a general focus on abstract layers which allowed specific platform implementations to be written for various modules as needed.

The core algorithms were based on existing work, with major contributions from Jules Bloomenthal and Paul Bourke for the polygonization, and Ken Perlin and Eric Heffert, as well as Larry Gritz for the ray tracing and raymarching.

Additional influences for the design were based on the software engineering elements presented by David Eberly.

The entire application was written in C++, and relies upon the OpenGL Utility Toolkit (GLUT) for the user interface. This reliance is only particular for this implementation, and the system was designed such that a different rendering component could be used.

POLYGONIZATION

The basic Marching Cubes polygonization relies on using precomputed look up tables to discretise a grid of values, based on their intersections with unit sized cubes.

Each possible state for every slice of a polygon can be determined by the whether a set of vertices pass through an edge.

This information can in turn be used to index into a geometry table describing the connections of a cube to perform the reconstruction.

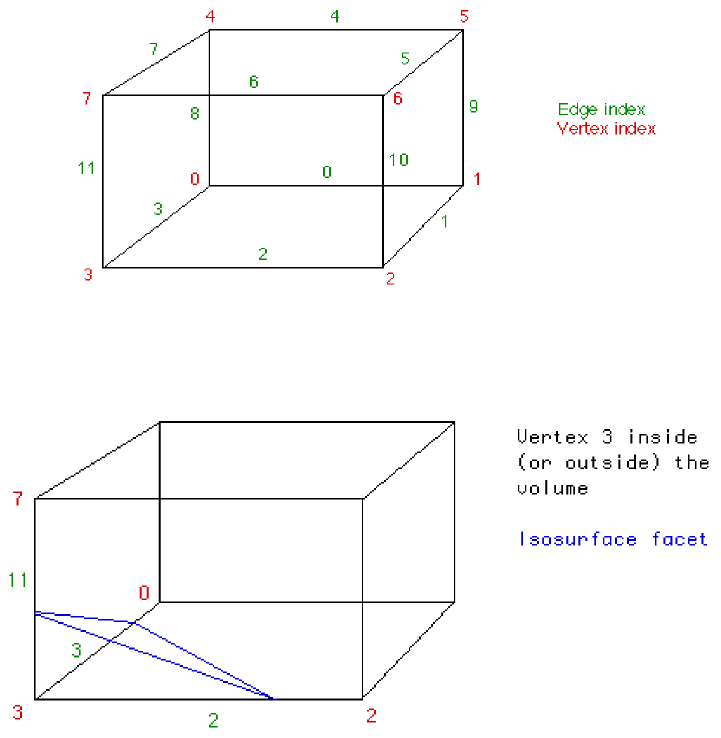


FIGURE 4 Basic indexing convention for cube marching an iso-surface (Bourke).

The tetrahedral polygonization is performed in exactly the same manner, except it splits a single cube into six tetrahedrons, and then uses the vertex positions and their relationship with the edges on a tetrahedron to determine index coordinates to create geometry.

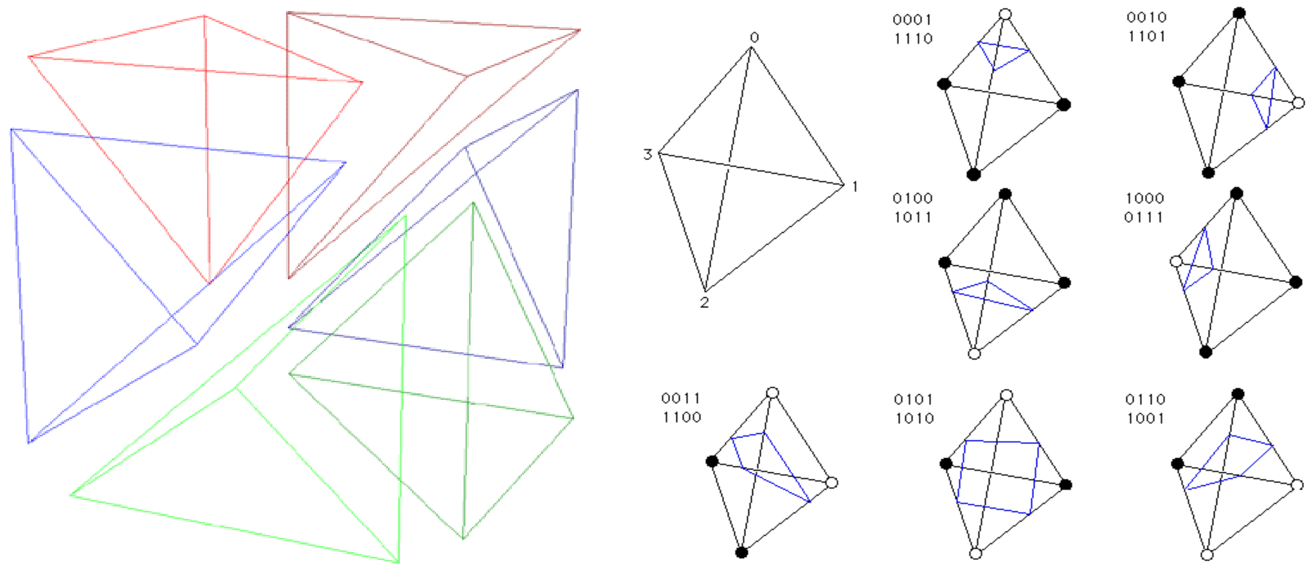


FIGURE 5 Tetrahedral decomposition of a unit cell and corresponding indexing convention (Bourke).

RAYTRACING AND HYPERTEXTURE

Since by definition *hypertexture* is a volumetric phenomena, it requires a volumetric rendering algorithm, which is a computationally expensive process.

Visualizer uses a ray marching ray tracer, which steps from front to back along each ray from the eye until total opacity is reached or the ray reaches a far clipping plane *(Ebert et al, pg 349)*. The basic algorithm is outlined in the following table.

This volumetric algorithm was then simplified to produce a standard ray traced visibility determination of an implicit. The surface was reconstructed by doing a simple bisection of the evaluation space and determining points of intersection.

```
The basic ray marching algorithm:

    Calculate a starting position
    Determine a starting segment to ray trace through
    While( current position is less than end position )
        Calculate the density at current position
        Calculate the light intersection point
        Calculate the opacity and color at current position
        Accumulate the current opacity and color
        Increment current position and setup for next iteration
    End
    Assign the final color from accumulated color and opacity
```

FIGURE 4 Basic ray marching algorithm (Apodaca, pg 416).

The first step to starting the ray march is calculating a starting position. The initial camera direction is used to fire a ray into the volume starting from the near clip plane and determine points of intersection within a bounding sphere, if they exist. Once an initial intersection point is determined, a ray march can begin along the viewing direction.

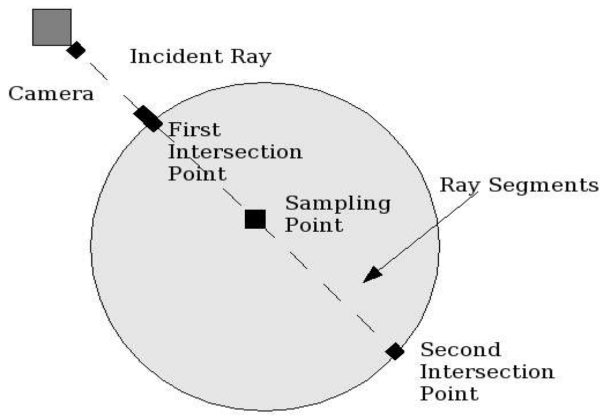


FIGURE 5 Determining intersection points along an incident ray (DG).

This is the technique employed by Perlin and Heffert, who used a fixed increment for stepping along the ray segment. In practice, this can lead to artifacts and aliasing due to undersampling if the step size is too small. This is minimized by jittering the initial step size for the ray march with a random offset, allowing small step sizes to be used without drastically reducing the quality of the visual texture.

As the ray is traversed, the DMF is sampled and tested. If the value falls within the range $0 < x < 1$, the current position lies within the soft region of the hypertexture volume and can be shaded.

A base color for the current position can be calculated by either using the density value itself, or using it as a variable to a function which determines an appropriate color value. Other methods for coloring include techniques which are based on position, or normalized surface directions.

If shading is required, a gradient direction can be calculated by sampling the DMF at three perpendicular locations within a small epsilon range to determine a normalized surface direction, or normal vector. If shading is not required, the base color can be used.

Once a normal vector is determined, it can be used for calculating diffuse and/or specular shading, as well as reflection and refraction within the volume. Typical shading functions include Lambertian (ideal diffuse), and Phong (empirical plastic-like).

In order to establish visibility, the opacity of the current position must be determined. This can be calculated using an approximation to atmospheric scattering. Perlin and Heffert derived a volume based function which was dependent on both density and step size. This also maintains the visual characteristics of the hypertexture when rendered with different step sizes. This is the approach used by Visualizer.

The resulting opacity is then applied to the current color value and accumulated. This is applied using traditional compositing methods by using additive blending.

Once the accumulation is complete, the final color contribution can be saved along with the alpha transparency, forming the final rendered image.

KNOWN ISSUES

As of this release, Visualizer has not been rigorously tested. This implies a certain level of instability, and in addition, there are several known issues.

Most notably, there are some discrepancies in the camera interpolation, and there are noticeable problems when the camera's near clipping plane gets too close to an object. This only applies to the ray traced and ray marched hypertexture renderers.

Additional issues include the inability to cancel a computation once a render has started, as well as some problems with hypertexture saturation.

REFERENCES

APODACA, A. A., 2000. *Advanced Renderman: Creating CGI for Motion Pictures.* Academic Press.

BOURKE. P. 2004. *Polygonizing iso-surfaces.* Available from: http://astronomy.swin.edu.au/~pbourke/modelling [Accessed 1 May 2004].

BLOOMENTHAL J. 1988. *Polygonisation of Implicit Surfaces.* Computer-Aided Geometric Design, 594) 341-355

EBERT, D., F. K. MUSGRAVE, D. PEACHEY, K. PERLIN, S. WORLEY, W. R. MARK, J. C. HART, 2002. *Texturing & Modeling: A Procedural Approach, Third Edition*. Morgan Kaufmann.

GUEZIEC, A., HUMMEL. 1995. *Exploiting Triangulated Surface Extraction using Tetrahedral Decomposition.* IEEE Transactions on Visualisation and Computer Graphics, 1 (4) 328-342

LORENSEN, W., CLINE H. 1987. *Marching Cubes: A High Resolution 3D Surface Construction Algorithm.* Computer Graphics, 21 (4): 163-169

PERLIN, K., HOFFERT E. 1989. *Hypertexture*, Computer Graphics (proceedings of ACM SIGGRAPH Conference); Vol. 23 No. 3.