Integrating Polygonal Objects on Voxel Based Terrain Models

LUIZ CARLOS CASTRO GUEDES

ADDLabs - Departamento de Ciência da Computação, Universidade Federal Fluminense
Praça Valonguinho s/n, Ed. Instituto de Matemática 4o andar, Centro, 24210-130 Niterói, RJ, Brasil
guedes@dcc.uff.br

Abstract. Voxel based projection is commonly used in applications that demand high exhibition frame rates of large scale terrain models. Placing tridimensional objects within a voxel based world may be an ease task for elementary objects. Complex objects are better represented by polygonal meshes. Integrating a voxel based terrain model with polygonal camera model brings together the benefits of both models. Such integration should be done with the aid of a z-buffer. In this paper we propose a solution to the problem of calculating the z-coordinate of each painted pixel in a voxel based terrain model. The proposed solution exploits the coherence of the pixels in the same column and uses a linear cost increment to update the z-coordinate. The proposed solution considers the situation when the view plane is not vertical.
Keywords: height fields, voxel projection, z-buffer, polygonal objects

1.Introduction

Large scale terrain models pose a heavy load on conventional polygonal models. Thus, ray casting based voxel projection is more adequate to the task because texture is placed into the model prior to exhibition and image driven algorithms avoid the overload of processing different indistinguishable information for each pixel [GuGaCa97].

Complex 3D objects are, on the other hand, better modeled by a polygonal mesh. Integrating polygonal objects on a voxel based terrain model should bring together the best of both worlds. Such integration should be done with the aid of a z-buffer. The terrain rendering algorithm should calculate the z-coordinate related to the observer of each painted pixel and store it in the z-buffer. Objects placed on the terrain should check and update the z-buffer during their rendering pipeline. Figure 1 shows a building rendered on voxel terrain borrowed from [SzGaCa97].

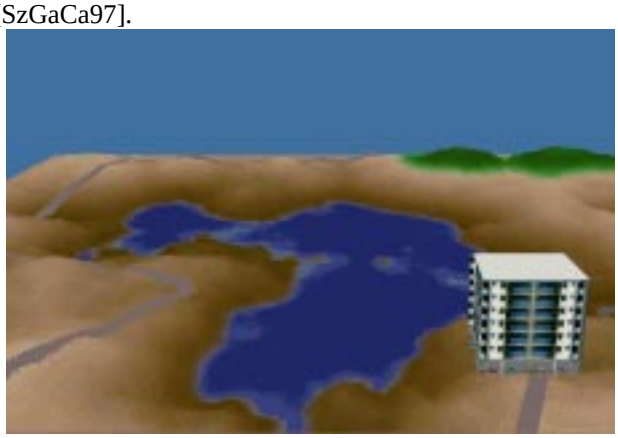

Figure 1 - Polygonal Object on a Voxel Terrain

The main issue when filling the z-buffer is to use an algorithm that exploits the coherence of the pixels of a column to find a linear cost increment for the z-coordinate from one pixel to the next.

In this paper we propose a solution to the problem of calculating the z-coordinate of each painted pixel in a voxel based terrain model. The proposed solution exploits the coherence of the pixels in the same column and uses a linear increment to update the z-coordinate from one pixel to the next and from one voxel to the next. The proposed solution considers the situation when the view plane is not vertical.

Section 2 presents the basic voxel algorithm. In Section 3 the solution to fill the z-buffer is proposed. In Section 4 some conclusions are taken.

2. Basic Voxel Algorithm

Each voxel has a height value, taken from the height field, and a color value, taken from a digital photograph. The presented approaches exploit the coherence between the pixels of a column to find linear cost incremental changes from one pixel to another.

The ray casting approach for terrain rendering assumes that the terrain is modeled by a Digital Elevation Map (DEM) and Digital Color Map (DCM). The DEM associates an elevation to each position (x,y) in the terrain and the DCM associates a color value to each position in the terrain. A column of the terrain raised with a height and color taken from the DEM and DCM, respectively, is called a *voxel*. Voxel algorithms consider the z coordinate as the altitude of the point instead of its depth. Hence, from now on, we will refer to the depth of a point as its **d** coordinate, and the buffer that stores the depth s the *d-buffer*.

Ray Casting algorithms may perform either forward casting or backward casting. Backward casting algorithms [LaMothe95] cast the rays that reach the farther distances before the ones that reach the closer distances.

The forward casting approach [Freese95] emits rays beginning from the observer to its projection on the ground and moving the destination of the rays far away on the ground looking for an intersection of a ray and the surface of the terrain. When an intersection occurs, the corresponding voxel is climbed up and the pixels of the screen are painted with its color. When the ray passes over the voxel the next position on the ground is inspected to find whether its voxel intersects with the ray.

In the case that the view plane is vertical, like in [SzGaCa97], the depth is the same for the pixels painted by each voxel and its value is just the distance **v** from the voxel to the observer. When the view plane rotates on the x-axis, many corrections have to be made to the generated image [GuGaCa97] and the depth is not so ease to determine.

3. d-buffer Filling Algorithm

To determine the depth of each pixel incrementally, we have to consider two distinct situation. The first one is when we climb up a single voxel painting the pixels of a screen column with the color of the voxel. The second situation is when the current voxel does not intersects with the current ray and we have to check the next voxel. Figure 2 illustrates both situations and shows the value of the depth increment (dd) in each case.

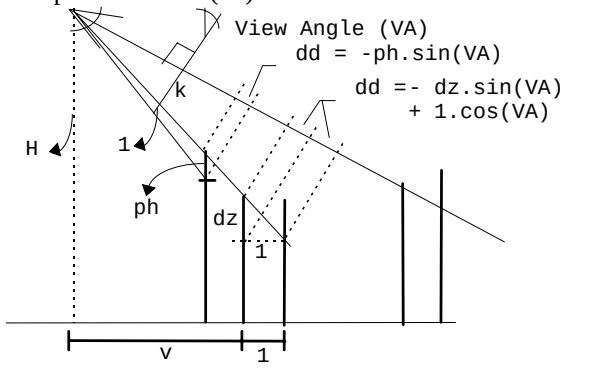

Figure 2 - Depth Increments

When we climb a voxel we have to increase the corresponding ray height by a value that corresponds to a lift of one single pixel on the screen. This value, called the pixel height (**ph**) may be determined from the slope of the two consecutive rays and the distance **v** on the ground from the current voxel to the observer.

The increment of the depth (dd) when we climb a voxel by ph corresponds to its projection on the viewing direction, which is given by **dd = -ph.sin(ViewAngle)**.

The increment of the depth (dd) when we go from one voxel to the next corresponds to the sum of the projections on the viewing direction of the unit step on the ground and the vertical fall corresponding to the unit step in the direction of the current ray, which is given by **dd= -dz.sin(ViewAngle) + 1.cos(ViewAngle)**.

Assuming that a ray starts at the observer, the initial depth value for each ray is zero. Algorithm 1 adapts one of the voxel projection algorithms of [GuGaCa97] to fill a d-buffer for a given column.

```
CastColumn( col, x0, y0, dx, dy, dz[] )
{
   z = H; x = x0; y = y0; // init position
   d = 0;                 // init depth
   row = MAX_ROW;         // start at last row
   for v = 1 to MaxRayDepth   {
      x +=  dx; y += dy; z += dz[row];
      d +=  cos(ViewAngle)-   // update depth
            dz[row]*sin(ViewAngle);
      h = GetHeight(x,y); // get voxel height
      if (h > z) { // ray intersects with voxel
         color=GetColor(x,y); //get voxel color
         do { // paint the pixel
            SetPixel(col,row,color);
            ph = v*(dz[row]-dz[row -1]);
            z += ph;   // update ray height
            d -= ph*sin(ViewAngle);//upd. depth
            dBuffer[row,col] = d;//upd.d-buffer
            if(--row < 0 ) return;
         }while(h>z); //ray intersects w. voxel
      }
   }
}
```

Algorithm 1 - Voxel Projection with d-buffer

4. Conclusion

We have presented a solution to the problem of filling a z-buffer during voxel based terrain rendering. The proposed solution exploits the coherence of the pixels in the same column to find linear increments to update the z-buffer. It has been implemented and successfully tested for under sea terrain with objects on it.

Acknowledgments

I would like to thanks to Ana Cristina Garcia for her unconditional faith on this work and to Rômulo Pinho for implementing algorithm presented in this paper.

References

A. La Mothe, *Black Art of 3D Game Programming*, Waite Group Press, 1995.

P. Freese, *More Tricks of the Game Programming Gurus*, Chapter 7, SAMS Publishing, 1995.

Guedes, L.C.C., Gattass, M., Carvalho, P.C., *Real -Time Rendering of Photo-Textured Terrain Height Fields*, SiBGraPI'97, 1997. <dpi.inpe.br/ambro/1998/06.01.14.19>.

Szenberg, F., Gattass, M., Carvalho, P.C., *An Algorithm for the visualization of a Terrain with Objects*, SiBGraPI'97, 1997. <dpi.inpe.br/ambro/1998/06.03.15.03>.