Volumetric Explosion Effects

Current explosion effects in games are typically created using a simple texture mapped polygon oriented parallel to the screen. This effect is easy to implement and is very fast in execution. However, the main problem with this billboard technique is that the explosion tends to ‘cut’ into other polygons in the scene (Fig. 1).

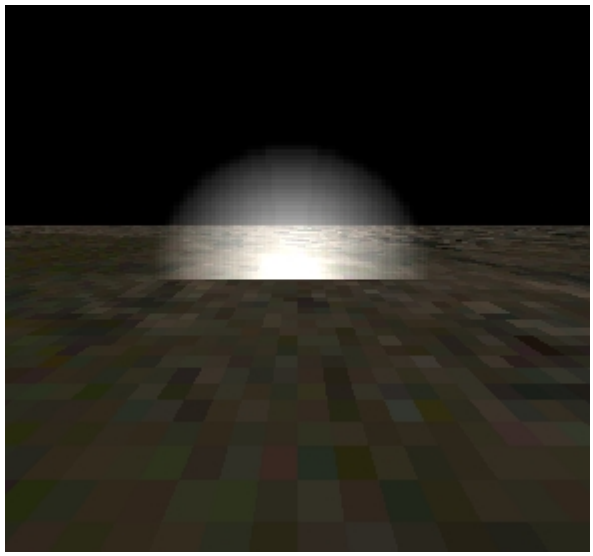

Fig. 1 Billboard explosion

This paper outlines a foundation for the generation of volumetric explosion effects (Fig. 2), which do not exhibit the usual artefacts of the billboard method.

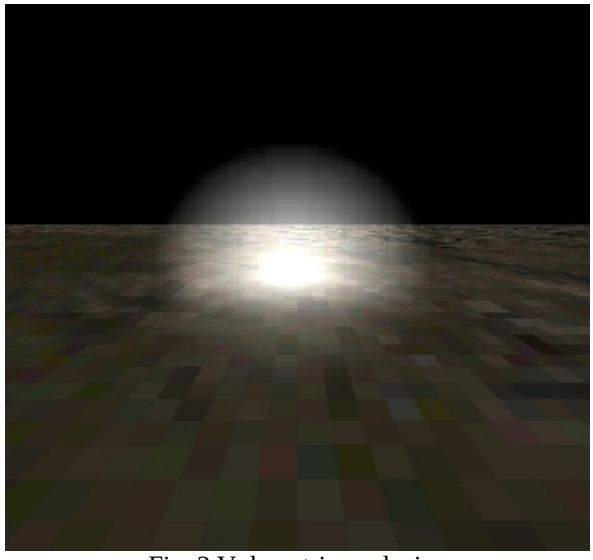

Fig. 2 Volumetric explosion

Overview of Volumetric Technique

The volumetric technique essentially treats the explosion as a well-defined volume of space. The premise is that any rendered polygon that is visible through the area of the explosion will have its appearance modified by the properties of the explosion. This includes polygons that are contained in the area of effect of the explosion, typically a sphere, and those that are behind the explosion as illustrated in Fig. 3.

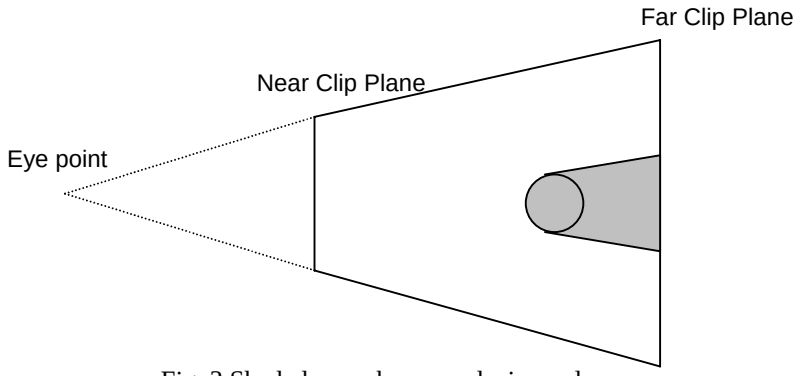

Fig. 3 Shaded area shows explosion volume

The visual effect is achieved by means of a second render pass over the polygons influenced by the explosion. The basic algorithm is as follows.

```
For every rendered polygon in scene do
        If polygon visible through explosion then
                Project explosion texture onto polygon
                Modulate intensity of polygon depending on distance from explosion
                Render polygon
        End If
End For
```

The following three sections deal with the visibility test, the texture projection, and the intensity modulation, each of which can be performed in either screen space or object space.

Visibility Test

The visibility test resolves to testing whether a polygon intersects with the required explosion volume. The explosion volume exactly the volume illustrated in figure 3. That is, a cone capped with a hemisphere. Performing this test exactly is unnecessary as explosion texture maps are generally created as roughly circular areas in a square bitmap.

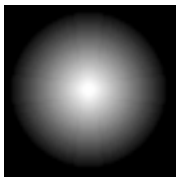

Figure 4: Explosion texture used in figures 1 and 2

This allows for the volume to approximated by a truncated pyramid similar to that used in typical view volume calculations, with the near clip plane defined as being at the point on the explosion sphere closest to the camera. The far clip plane is the same as the far clip plane of scene. If the visibility test is carried out in

world space then each polygon must be checked to see if it lies within the volume defined by the six planes. Carrying out the test in screen space is a much simpler process as the projection of the explosion volume results in a circle (if using the conical volume) or a square if using the truncated pyramid. Note that if the projection results in a pixel aspect ratio not equal to 1:1 then the projected areas would be an ellipse and a rectangle respectively.

Texture Projection

Once a polygon is deemed to be affected by the explosion, it must then have the explosion texture projected onto it. This takes the form of a basic planar texture projection along the direction vector of the camera. If this projection happens in world space, then the simplest method is to transform the vertices of the triangle into view (camera) space, and perform the projection there. Note that the effective radius of the explosion has to be scaled to accommodate for perspective foreshortening effects. This is scaling is dependant on the distance from the vertex to the eye point.

Again, performing the calculation in screen space is much simpler, as the scaled radius of the explosion need only be calculated once.

The equations for this calculation are,

$$u = 0.5 + (v_x - c_x)/(2 * Rad)$$
$$v = 0.5 + (v_y - c_y)/(2 * Rad)$$

where u and v are texture co-ordinates,
c is the centre of the explosion (in screen space)
v is the current vertex (in screen space)
Rad is the scaled radius of the explosion

This will generate values in the range [0..1] for points in the explosion volume. For points outside the explosion values <0 and >1 are generated. This can be compensated for by either clipping the polygons to the edges of the volume (not recommended as this may introduce rendering inconsistencies), or by setting the texture wrapping mode to a clamp setting. In this mode a pixel whose texture co-ordinates lie outside the range [0..1] is set to the colour at 0 or 1 respectively. Clipping only becomes necessary for those polygons whose texture co-ordinates are outside of the maximum range supported in the rendering API (Direct3D requires texture co-ordinates to be in the range [-128..+127).) Clipping must also be performed on those polygons that extend from in front of the eye point to behind the eye.

Intensity Modulation

This stage modifies the intensity of the explosion effect, depending on the distance between the vertex and the centre point of the explosion. For efficiency, it is assumed that any intensity modulation in the screen x and screen y directions is handled by the intensities in the texture map itself. All we need is a scheme where we modulate according to the distance of a vertex along the screen z-axis. This can be done in two ways. The first is to assert that any point further from the camera than the centre of the explosion is deemed as being full intensity. The second is to assert that any point further from the camera than the rear of the explosion is full intensity. Whichever scheme is chosen is dependent on the application. Note that the second is more ‘correct’ although the first has better optimisation implications (see next section). The resulting calculation can be performed as a simple linear interpolation, although this does not accurately take into account perspective effects. Just as before, polygons must be clipped to achieve correct effects. The polygons must be clipped to the planes of maximum and minimum intensity (i.e. the front plane of the explosion and either the centre or back plane).

Note that the intensity calculation is applied differently depending on whether an additive or modulating blend is used in the application of the second render pass. In additive blending, the colour of the vertex is varied between white and black, under modulation the alpha component is altered.

Optimisations

There are two basic ways to optimise the implementation of this technique. The first is to perform the various calculations more quickly, the other is to perform fewer calculations!

The algorithm is more easily and efficiently implemented in screen space. This means that all geometry is transformed a single time, and then post processed to produce the explosion effects. No further transformations are necessary (other than transforming some reference points on the explosion volume to screen space). There are potential problems with this method, which are outlined in the next section.

The second class of optimisation can be achieved by using a hybrid algorithm. This places a billboard explosion at the plane of maximum intensity in the explosion volume. If the centre of the explosion is deemed as having maximum intensity, then a billboard is placed at the centre of the explosion. This allows us to limit our explosion volume to only that space between the front of the explosion and the billboard.

Clipping issues

A number of visual artefacts become visible when polygons are clipped to the explosion volume. As explained previously, this clipping is necessary to achieve correct results for intensity calculations and to prevent the generation of texture co-ordinates that are too large. Clipping also reduces the area of the screen that has to be re-rendered. The main problem with clipped polygons is that z-buffer fighting is introduced. The most obvious solution (other than not clipping) is to use z-bias functionality in the API to allow us to render overlaid polygons correctly. Unfortunately, z-bias is not currently widely supported. The same effect can be achieved by applying a translation in z, after the generation of screen space co-ordinate. This can be achieved by simply modifying the projection matrix as follows. (Note that Direct3D requires a projection matrix to be in the format below). The z-bias required is dependent not only on the resolution of the z-buffer, but also potentially on the hardware.

$$\mathbf{M}_{\mathbf{proj}} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & d \\ 0 & 0 & e & 0 \end{pmatrix}$$ - Projection Matrix

$$\mathbf{M}_{\mathbf{tran}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \delta & 1 \end{pmatrix}$$ - Z Translation Matrix

$$\mathbf{M}_{\mathbf{projbias}} = \mathbf{M}_{\mathbf{proj}} \bullet \mathbf{M}_{\mathbf{tran}} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c + \delta d & d \\ 0 & 0 & e & 0 \end{pmatrix}$$

Limitations

A number of limitations were found whilst implementing the algorithm. The most severe of these came from implementing the screen space variant under Direct3D. The IDirect3Ddevice::ProcessVertices method was used in conjunction with vertex buffers to read back screen space co-ordinates. This presented problems, as access is required to all screen space vertices and polygons, including those generated by clipping algorithms. Put simply, effective screen space implementation requires that the application performs it's own transformation and lighting calculations.

A second limitation is that under the pure world and screen space implementations it was necessary to ensure that the entire region visible through the explosion was previously rendered to with polygons. This is because the explosion pass would not show on areas of the screen filled by a flat colour (such as a screen clear) – the result was as shown in figure 5.

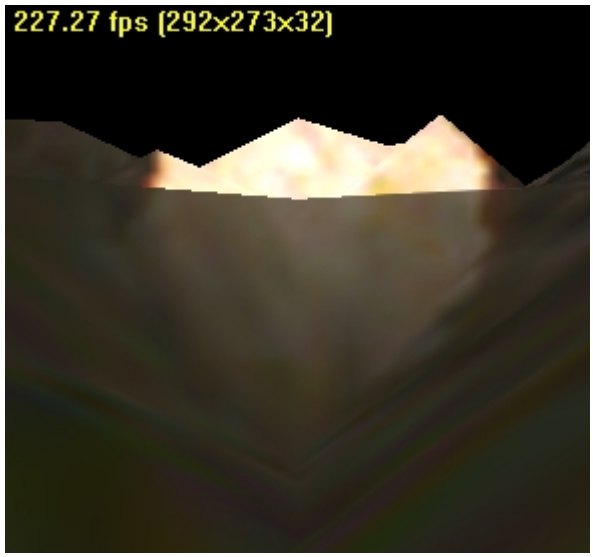

Figure 5. One limitation of the 'pure' method

It should be noted that, due to the multi-pass nature of this technique, it is possible for any number of explosions to intersect, thus producing a cumulative effect. The drawing order becomes an issue when using a modulation blend mode rather than additive.

Future Implementations

This technique effectively emulates planar projected texture mapping at the application level. The application could be modified to use these features in an API that provides direct support. This would potentially eliminate the need to perform clipping and hence remove the need for z-biased rendering. It is also conceivable to use textures for both the explosion graphic and the intensity. The intensity map would be a simple gradient texture that could be projected vertically i.e. along the negative screen space y axis. The explosion detail would be projected along the screen z-axis. This would allow the use of this technique in systems that use w-buffering and would eliminate most of the implementation 'curiosities', as well as improving performance dramatically.