

eisblogs

Billboarding on the GPU

June 19, 2005

Screen-aligned billboarding is to rotate a quad to always face the camera. The standard way of computing this transformation is to compute an orthogonal basis out of the `eye_billboardcenter` vector and an up-vector.

Let the GPU do the work

A quad has 4 points, so we pass 4 vertices at the center position per billboard. Now we have to find out which one of the 4 vertices belongs to which corner of the billboard. There are more memory friendly ways but lets just use 2d texcoords for this. The upper left point gets uv-coords (-0.5,0.5), upper right (0.5,0.5), lower right (0.5,-0.5), lower left (-0.5,-0.5).

Now we compute our ortho-basis in the vertex program resulting in an normalized "up" and a normalized "right" vector. The "up" vector is now multiplied with the v part of our texcoords, the "right" vector with the u component. By adding these two scaled vectors to our position we get a quad. Now this requires quite alot of math and 2x normalizing building the square-root in the ARBvp. The GPU is fast but we can do better, can't we?

Make things more effective

Let's recap what billboards are: screen aligned quads (we talk about spherical billboards here, axis aligned work similar). Now if we just could pull out a quad of the 4 points when they are already on-screen. We can! In eye-space everything but the projection is applied to the vertices. In OpenGL the x/y plane is parallel to the screen. So we could just apply our offsetting to x and y in eyespace to get our screen-aligned quad and do the projection afterwards. The code for this technique is below. All one have to do is to separate the ModelViewProjection transform into ModelView transform, offsetting and then projection.

OpenGL ARB_vertex_program source

```
PARAM mv[4] = { state.matrix.modelview[0] }; #modelview matrix
PARAM prj[4] = { state.matrix.projection };
```

```
TEMP pos,projpos;
```

```
#####
# Apply modelview transformation
DP4 pos.x, mv[0], vertex.position;
DP4 pos.y, mv[1], vertex.position;
DP4 pos.z, mv[2], vertex.position;
```

```
DP4 pos.w, mv[3], vertex.position;

#####
# Add offset
ADD pos.xy, pos, vertex.texcoord[0];

#####
# Apply projection matrix
DP4 projpos.x, prj[0], pos;
DP4 projpos.y, prj[1], pos;
DP4 projpos.z, prj[2], pos;
DP4 projpos.w, prj[3], pos;

MOV result.position, projpos;
```

Powered by [bBlog](#)
Based on a design by [Michael Heilemann](#). Ported to bBlog by [Raefer Gabriel](#).