CPSC 414, Project 2: Paintball++

Out: Wed 22 Oct 2003
Due: Tue 4 Nov 2003 5pm PST
Value: 10% of final grade

Your favorite elephant is sick. To work out your irritation about this sad state of affairs, you want to play Paintball++ with your Big Fungible Gun (BFG). Of course, first you have to implement it: that's the project. There are five required parts (50 points), and up to 5 more extra credit points can be earned.

Cubes: [2 pts] Your Paintball++ world contains 20 objects inside the cube $-50 < x, y, z < 50$. These objects are placed at random inside the volume. By default, the objects are cubes. Their size should be random within the range of diameters between 1 and 5. Hitting 'n' should give you a new neighborhood of objects (the old ones deleted, the new ones at new random positions). Hitting 'e' should erase any graffiti you blasted as below (lines or textures).

Flying: [10 pts] You should implement camera movement where you control forward/backward speed, roll, pitch, and yaw with the mouse. (Pitch is rotation around the side-to-side axis, like nodding your head up and down for yes. Yaw is rotation around the vertical axis, like shaking your head from side to side for no. Roll is rotation around the front-to-back axis, like tilting your head so your ear touches your shoulder (like the up vector in the lookat model). There's a good animated illustration at http://www.nasm.si.edu/galleries/gal109/NEWHTF/PITCH.HTM

Hitting 'r' should reset the camera to the origin of the neighborhood cube. Your main movement is through mouse drags. Dragging vertically with the left mouse controls your forward or backward speed. Consider the size of the vector from the drag start to drag end, the bigger the the vector the higher the speed. Up is forward, down is backward. Dragging horizontally controls your roll turn angle: right is a roll to the right, and left is a roll to the left. Again, the size of the vector controls the amount, with bigger equivalent to more. The middle mouse similarly controls yaw with horizontal drags and pitch with vertical drags. All of this motion is with respect to the current view. You'll use these camera controls to fly around the scene in order to blast the objects. You'll need to experiment with how to set the weights when converting drags to motions, in order to make flying feel natural.

Picking and Lineblasting: [10 pts] Hitting 'l' causes your BFG to shoot a pick ray into the scene at the current cursor position, and where the ray intersects the nearest surface you should draw a rectangle centered around the pick point and on the plane tangent to the intersected surface. (Actually, if you drew the rectangles exactly in the tangent plane, they wouldn't be visible when the object is flat - so draw them a little bit further out than the tangent plane, along the direction of the normal, so that they aren't drawn in exactly the same place as the surface. If the cube is directly facing the screen, you'd creep a little bit forwards in z. You may use OpenGL support for picking. The rectangle should be 3 pixels wide and 3 tall in screen space when created. Of course, when the camera moves the screen size of the rectangle will change. You should specify its position in world coordinates such that it's 3x3 in screen coordinates given the viewpoint when the pick occurs. So if you hit a very far-off object, when you fly close to look at it in more detail your rectangle will be quite big with respect to the object size. If you get so close that the object fills the window, you'll have small rectangles with respect to the object. So your BFG acts similar like an airbrush: the paint spreads out more the further it gets from the nozzle of the paintgun.

When you hit 'l' a second time on the same surface in another spot, you should draw a second rectangle centered around the new pick point, again in the plane tangent to the object surface. Then your BFG will blast a line on the plane between this pair of pick points, doing its own scan conversion. It should draw your line across the same plane picked face of the cube using a series of these 3x3 rectangles as your "pixels" - that is, the building blocks of the line. You **cannot** use an OpenGL line: you need to scan convert the line yourself. Bresenham's line scan conversion algorithm would be a very good starting point for this! When you hit a surface the third time, you start a new line. In general, every odd hit starts a line, every even hit ends a line. Once you intersect a new polygon, the hit count starts over. So you could just have a single rectangle marking a polygon, if you don't hit it again immediately. You can draw shapes with your BFG, if you're feeling artistic and you're close enough to the object to have enough room. If you're really far away, you probably can't draw any lines, because it will be hard to hit two different spots on a cube face if it's only a few pixels across.

Extra credit: [3 pts] Antialiasing. Antialias your line.

Textureblasting: [15 pts] When you hit 't', you again blast a pick ray to find the frontmost surface, and your BFG textures the object with the current image loaded into it. You can change your textureblasting settings by hitting a digit to control the number of tiles to use. The default is 1 tile per cube face. You should support from 1-9 tiles per cube face or sphere. Use 'i' to switch to the next image loaded into your BFG. You should have at least 3 different images, starting with the default of a checkerboard that's 5 boxes across and 5 boxes down. (You can download this default texture from the web site.) It's OK to hardwire the image file names into your code.

Spheres: [3 pts] Use spheres instead of cubes: hit 's' to switch to sphere mode (this is like hitting 'n', in that a new universe is generated). You do not need to implement lineblasting for your spheres, but you do need to support textureblasting. The texture default should be 1 tile for the entire sphere. Again, support up to 9 tiles.

Projection: [10 pts] Realism is so obvious, Paintball++ has more options! Hit 'p' to toggle between the standard perspective projection with a 60° field of view and cabinet projection mode. Your initial oblique angle should be 15°. 'O' increases the angle of obliqueness by 1 degree, 'o' decreases it. (Make sure it never goes below 2 or above 88.)

Extra credit: [2 pts] Bump Mapping. Have your blasted line be bump mapped, so that the original two points and the line are shown by rectangular bumpmapped indentations in the surface rather than as new rectangles drawn just in front of the object. The bumpmapped rectangles should be 2diameter big. Hit 'b' to toggle bumpmap mode. (It should be off by default.)

Suggested Strategy

Start with generating cubes, then do flying, then do picking to generate a single rectangle. Once that's solidly tested and working, move on to the scan conversion to create the whole line. Then do texturing for a single tile with the cube. Then implementing tiling for the cube. Once all of that is working, move on to spheres. The projection can be whenever you want, but it's probably good to get a pretty solid basic set of functionality first. Don't start on extra credit until you have everything else done.

Handin

Do the same thing as with project 1, except: use the command 'handin cs414 proj2', and only bring hardcopy of the README file to your demo slot (that is, no need to bring code printouts). We will again be grading through face-to-face demos. The best work will be posted on the course web site in the Hall of Fame.