GAME DEVELOPMENT > PROGRAMMING

Let's Build a 3D Graphics Engine: Dynamic Lighting

by Kyle Sloka-Frey 26 Jun 2013

Difficulty: Intermediate Length: Short Languages: English

Programming Platform Agnostic HTML5

This post is part of a series called Let’s Build a 3D Graphics Software Engine.

Let's Build a 3D Graphics Engine: Colors

Hello! This is the final article in our series on the basics of 3D software graphics systems, and this time we will be looking into dynamic lighting! Before you get too excited or too worried about how difficult a task like dynamic lighting can be, relax. We are only going to be covering the most basic form of dynamic lighting for now (since the entire subject itself is huge, and others have managed to fill entire books on the concept).

Specifically, we will be building a single point, single color, fixed lighting radius dynamic lighting system that will allow us to start dabbling in the subject. Before getting into that, though, let's take a look at some of our previously made classes that we will be using.

Recap

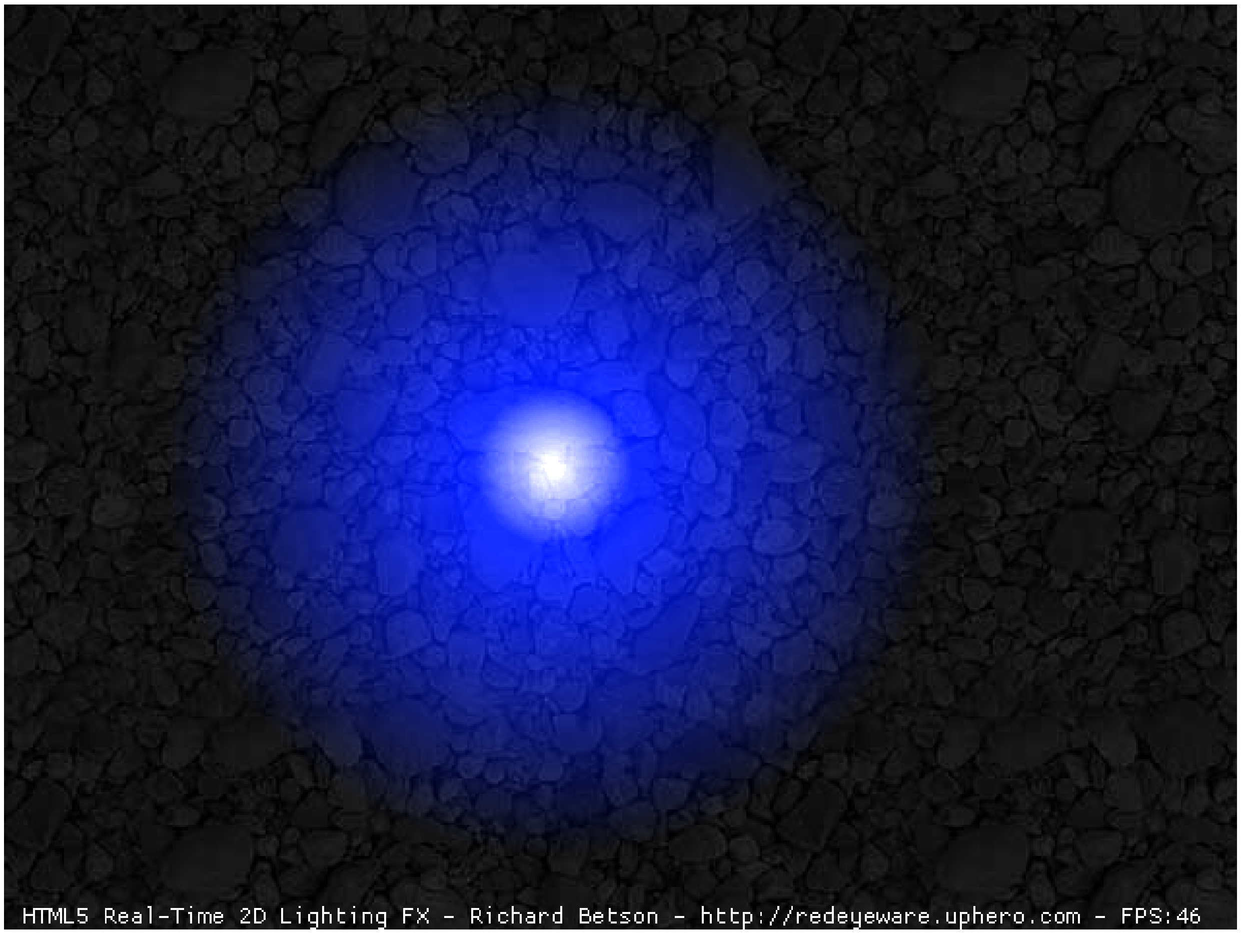
Our dynamic lighting is going to be handled on a point-by-point basis as they are prepped to be drawn to the screen. This means that we will be making extensive use of two of our previous classes: the `Point` class and the `Camera` class. Here's what they look like:

```
Point Class
{
    Variables:
        num tuple[3]; //(x,y,z)
    Operators:
        Point AddVectorToPoint(Vector);
        Point SubtractVectorFromPoint(Vector);
        Vector SubtractPointFromPoint(Point);
        Null SetPointToPoint(Point);
    Functions:
        drawPoint; //draw a point at its position tuple
}

Camera Class
{
    Vars:
        int minX, maxX;
        int minY, maxY;
        int minZ, maxZ;
        array objectsInWorld; //an array of all existent objects
    Functions:
        null drawScene(); //draws all needed objects to the screen
}
```

Using this info, let's put together our basic lighting class.

Our Lighting Class


HTML5 Real-Time 2D Lighting FX - Richard Betson - http://redeyeware.uphero.com - FPS:46

An example of dynamic lighting. Source: http://redeyeware.zxq.net

Our lighting class is going to need a few things to make it functional - namely a position, a color, a type, and an intensity (or lighting radius).

As I said earlier, our lighting is going to be calculated on a point by point basis prior to each point being drawn. The upsides of this are that it is simpler for how we have our engine organized, and also that it shifts more of our program's load to the system's processor. If you were to pre-calculate your lighting, it would instead shift the load to your system's hard drive, and depending on how your engine is designed, may be simpler or more complex.

With all of this in mind, our class could look like this:

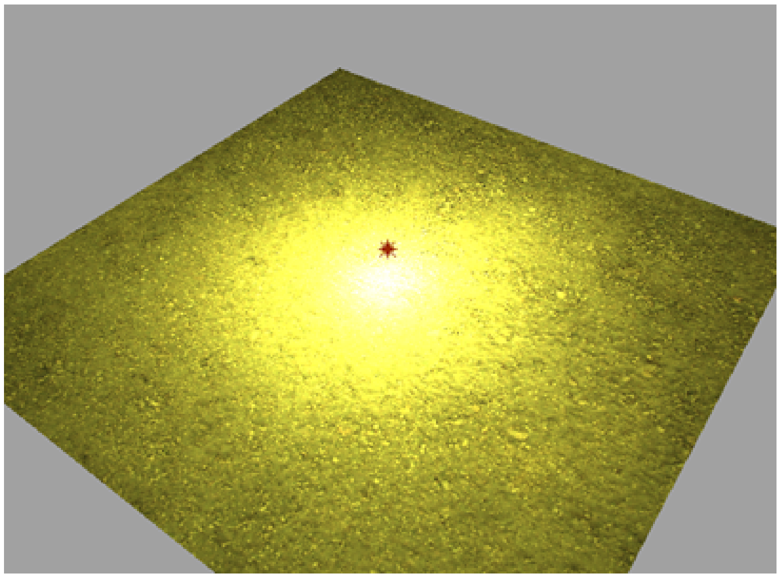
```
Lighting Class
{
    Variables:
        num position[3]; //(x,y,z)
        num red = 255; //what to add to a point's r value at full intensity
        num green = 255; //what to add to a point's g value at full intensity
        num blue = 255; //what to add to a point's b value at full intensity
        string lightType = "point"; //the type of lighting
        num radius = 50; //the light's radius in pixels
}
```

Currently we are going to leave all of its values hard-coded in for simplicity, but if you find that you want to expand the lighting classes functionality, you could easily have each of the values be modifiable through either functions, a constructor, etc.

All of the important math for our dynamic lighting is going to be happening within our camera class though, so let's take a look at that.

Advertisement

Lights? Camera? Engine.



Another example of dynamic lighting. Source: http://blog.illuminatelabs.com/2010/04/hdr-and-baked-lighting.html

We are going to add a new variable to our camera class now, which we'll use to store our light source. For now, the variable will only be storing one lighting instance, but it could easily be scaled up to allow for more points of light.

Right before a point is drawn, we are going to check to see if it is within the radius of our light. Once we know that it is within the light's range, then we need to find the distance between the point and the light's position, and then we need to adjust the point's color based on that distance.

With all of that in mind, we can add code similar to this to our camera's `drawScene()` function:

```
if(currentPoint.x >= (light.x - light.radius)){ //if the point is within the light's le
    if(currentPoint.x <= (light.x + light.radius)){ //if the point is within the light'
        if(currentPoint.y >= (light.y - light.radius)){ //if the point is within the li
            if(currentPoint.y <= (light.y + light.radius)){ //if the point is within th
                //calculate distance between point and light (distance)
                //calculate percentage of light to apply (percentage = distance / radiu
                point.red += (light.red * percentage); //add the light's red, scaled to
                point.green += (light.green * percentage); //add the light's green, sca
                point.blue += (light.blue * percentage); //add the light's blue, scaled
            }
        }
    }
}
```

As you can see, our method of adjusting a point's color currently isn't all that advanced (there are many that are though, if you wish to use them instead). Depending on a point's distance from the center of the light, we lighten its color by a percentage. Our lighting method does not account for shading at all, currently, so areas far away from the light won't become darker, and objects won't block light from other objects that may be behind them.

Follow the Light

For our program this time, we are going to have a few pre-rendered shapes on the screen. These can be anything that you want, but for our example, I'm just going to use some simple points. Now, when a user clicks anywhere on the screen, we are going to create a light source at that point. The next time that they click, we will move the point to that position, and so on. This will allow us to see our dynamic lighting in action!

Here's what your program might look like:

```
main{

    //setup for your favorite Graphics API here
    //setup for keyboard input (may not be required) here

    var camera = new Camera(); //create an instance of the camera class

    //set the camera's view space
    camera.minX = 0;
    camera.maxX = screenWidth;
    camera.minY = 0;
    camera.maxY = screenHeight;
    camera.minZ = 0;
    camera.maxZ = 100;

    //draw initial objects and set them to the camera space

    while(key != esc) {
        if(mouseClick) {
                if(firstClick) {
                        //create the initial light object at mouse pos
                }
                else {
                        //modify the light object's position
                }
                camera.drawScene();
        }
    }
```

Now you should be able to experience your dynamic lighting in action, and hopefully see how much more depth it can add to a game engine. Check out my demo here - use the **A** and **S** keys to scale, and the **Y**, **H**, **U**, **J**, **I** and **K** keys to rotate.

Conclusion

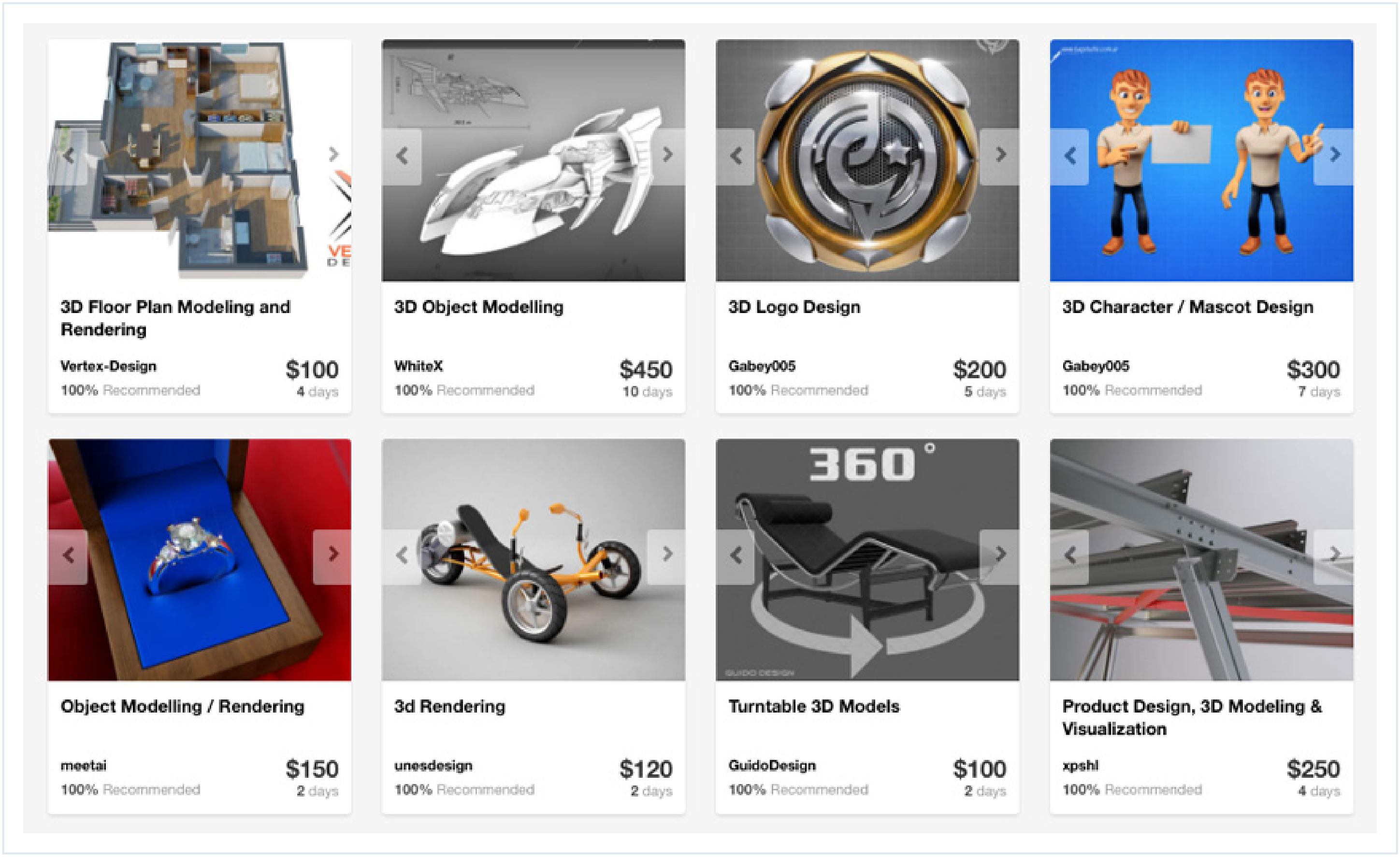
While our dynamic lighting is simple, it can certainly be improved if you feel inclined to do so. Some things that would be pretty cool, and also pretty simple to add in are:

- adjustable lighting radius
- adjustable lighting color (instead of lightening a color uniformly, lighten it by a fraction of its set color)
- block light from traveling past solid objects
- handle multiple points of light
- add a shadow to all points outside of the light's radius

- experiment with other forms of light (directional, cone, etc.)

Thank you for checking out our series, Let's Build a 3D Game Engine. It's been great writing these articles, and remember, if you have any questions, please feel free to ask them in the comments below!

You can also get extra help over on Envato Studio, where you can find plenty of fantastic 3D Design & Modeling services for affordable prices.



3D Floor Plan Modeling and Rendering
Vertex-Design $100
100% Recommended 4 days

3D Object Modelling
WhiteX $450
100% Recommended 10 days

3D Logo Design
Gabey005 $200
100% Recommended 5 days

3D Character / Mascot Design
Gabey005 $300
100% Recommended 7 days

Object Modelling / Rendering
meetai $150
100% Recommended 2 days

3d Rendering
unesdesign $120
100% Recommended 2 days

Turntable 3D Models
GuidoDesign $100
100% Recommended 2 days

Product Design, 3D Modeling & Visualization
xpshl $250
100% Recommended 4 days

3D Design & Modeling services on Envato Studio

Advertisement