

# Rendering Lights

## Video

[http://www.leadwerks.com/files/Tutorials/CPP/Rendering\\_Lights.wmv](http://www.leadwerks.com/files/Tutorials/CPP/Rendering_Lights.wmv)

## Writing the Program

We start with a basic program that displays a spinning cube:

```
#include "engine.h"

int main(int argc, char** argv)
{
    Initialize();

    // Create a graphics window
    Graphics(800,600);

    // Create a world
    if (!CreateWorld())
    {
        MessageBoxA(0,"Error","Failed to create world.",0);
        goto exitapp;
    }

    // Create a camera
    TEntity cam = CreateCamera();
    MoveEntity (cam, Vec3(0,0,-5) );

    // Create a visual mesh
    TEntity mesh = CreateCube();

    // Main program loop
    while(!KeyHit(KEY_ESCAPE))
    {

        // Make the visual mesh spin
        TurnEntity (mesh, Vec3(0.5f));

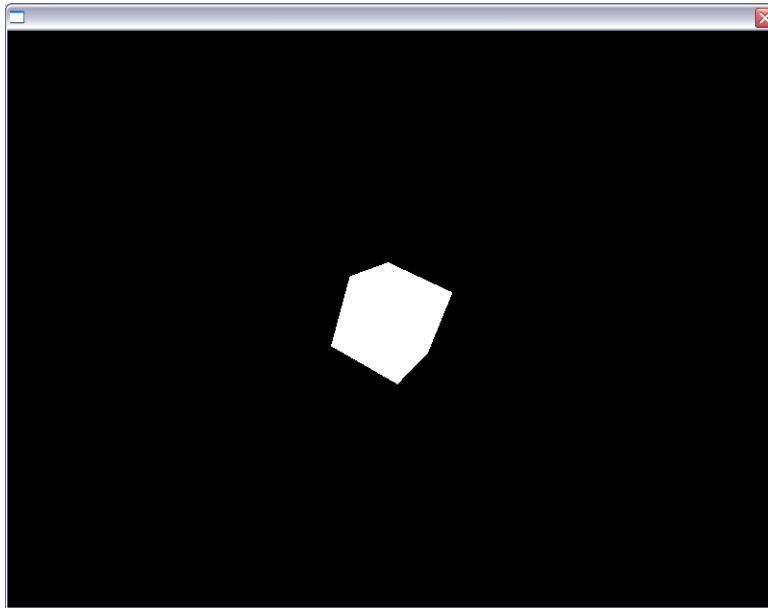
        // Update timing and physics
        UpdateWorld();

        // Render the world
        RenderWorld();

        // Swap the buffers so we can see what was drawn
        Flip ();
    }
}
```

```
exitapp:  
  Terminate();  
  return 0;  
}
```

Notice this program does not display any lighting, and the cube is pure white:



### Adding a Light

It's easy to add a light to the program. We can choose from point, spot, or directional lights. Let's create a spot light and position and rotate it so it shines on the cube. Add this code before the main loop:

```
TLight light=CreateSpotLight();  
PositionEntity(light,Vec3(2,2,-2));  
RotateEntity(light,Vec3(45,45,0));
```

However, we won't be able to see the effects of the lighting until we do a few more things.

## Deferred Lighting

Starting in Leadwerks Engine version 2.1 a deferred renderer is used. This treats lighting as a post-processing step and only renders lights on the final screen pixels, after everything is drawn. As objects are drawn, the colors, depth values, and normals are stored in a set of textures called a *render buffer*. Lights are then drawn using the data from the render buffer, as in this diagram:

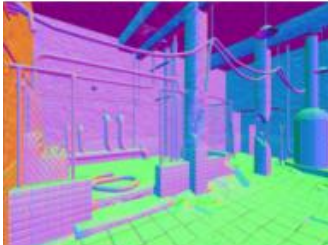
Albedo



Depth



Normal



Specular factor



## Creating a Render Buffer

To render lighting we must create a render buffer with color, depth, and normal components. Add this code before the main loop:

```
TBuffer buffer=CreateBuffer(800,600,BUFFER_COLOR0|BUFFER_DEPTH|BUFFER_NORMAL);
```

## Rendering Lights

Our rendering code is going to look a little different now. First we use the SetBuffer command to tell the engine we want to draw to the render buffer we created. Then we call RenderWorld to draw the world. We then set the buffer to the back buffer, which is the default render buffer the engine uses. Finally, we pass our render buffer to the RenderLights command. This will read the data we wrote to the render buffer and draw it onto the current buffer, which in this case is the back buffer.

```
//Make our render buffer the current buffer
SetBuffer(buffer);

//Render the world to the render buffer
RenderWorld();

//Make the back buffer the current buffer
SetBuffer(BackBuffer());

//Call the RenderLights command, passing our buffer which contains
//color, depth, and normal data
RenderLights(buffer);
```

Let's also add another object for the cube to cast shadows onto. Add this code before the main loop:

```
TMesh ground=CreateCube();
ScaleMesh(ground,Vec3(10,0.1,10));
PositionEntity(ground,Vec3(0,-2,0));
```

Our final program looks like this:

```
#include "engine.h"

int main(int argc, char** argv)
{
    Initialize();

    // Create a graphics window
    Graphics(800,600);

    // Create a world
    if (!CreateWorld())
    {
```

```

        MessageBoxA(0,"Error","Failed to create world.",0);
        goto exitapp;
    }

    TBuffer buffer=CreateBuffer(800,600,BUFFER_COLOR0|BUFFER_DEPTH|BUFFER_NORMAL);

    // Create a camera
    TEntity cam = CreateCamera();
    MoveEntity (cam, Vec3(0,0,-5) );

    // Create a visual mesh
    TEntity mesh = CreateCube();

    //Create another mesh to cast a shadow on
    TMesh ground=CreateCube();
    ScaleMesh(ground,Vec3(10,0.1,10));
    PositionEntity(ground,Vec3(0,-2,0));

    //Create a spotlight
    TLight light=CreateSpotLight();
    PositionEntity(light,Vec3(2,2,-2));
    RotateEntity(light,Vec3(45,45,0));

    // Main program loop
    while(!KeyHit(KEY_ESCAPE))
    {

        // Make the visual mesh spin
        TurnEntity (mesh, Vec3(0.5f));

        // Update timing and physics
        UpdateWorld();

        //Make our render buffer the current buffer
        SetBuffer(buffer);

        //Render the world to the render buffer
        RenderWorld();

        //Make the back buffer the current buffer
        SetBuffer(BackBuffer());

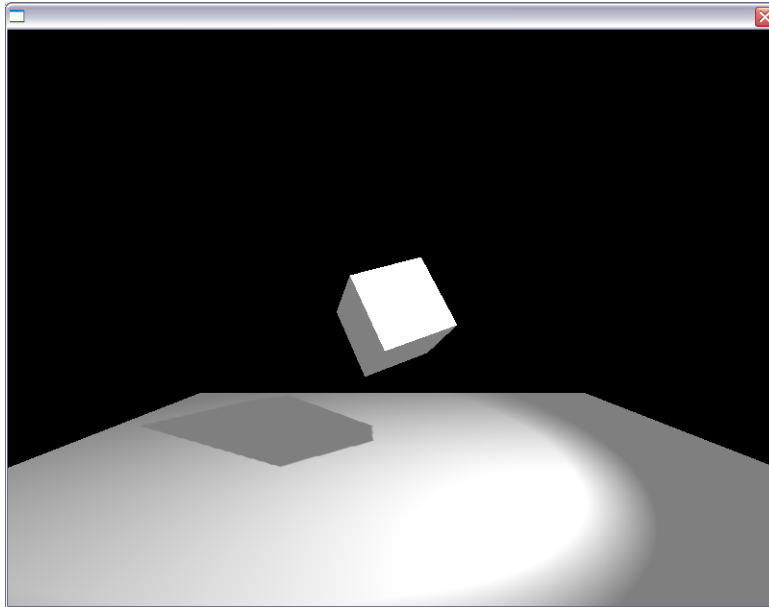
        //Call the RenderLights command, passing our buffer which
        //contains color, depth, and normal data
        RenderLights(buffer);

        //Swap the buffers so we can see what was drawn
        Flip ();
    }

    exitapp:
    Terminate();
    return 0;
}

```

When we run the program, we will see a spotlight shining onto the cube and casting a shadow onto the ground below:

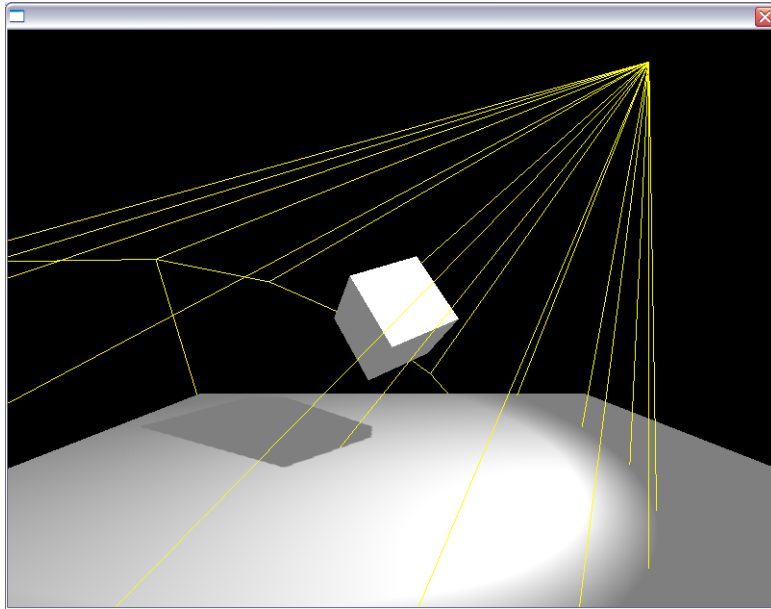


### Visualizing Lights

The command `DebugLights` can be used to draw a visual representation of a point or spot light's 3D volume. Add this line anywhere in your source code and run the program:

```
DebugLights(true);
```

Here is the result:



#### Further Reading

- [Deferred Lighting in Leadwerks Engine](#)

Copyright Leadwerks Software 2008

All Rights Reserved.