

Shadow Code Integration

The integration should be simple if you want generic shaders, and non of the texture and uniform resource conflict..

1. Include ShadowWrapper.h in the application file.
2. Instantiate and configure the wrapper in the Application::Configure virtual method after the scene graph has been initialized like this:

```
RefPtr<ShadowWrapper> mShadowWrapper;  
mShadowWrapper = new ShadowWrapper();  
mShadowWrapper.get()->Config(  
    mInfiniteLight,  
    GetScene(),  
    GetCamera()->GetCamera(),  
    1,  
    true,  
    2048,  
    1.3f,  
    0.2f  
);
```

You will also want to create an infinite light, and tell the application not to use the default light source. Like this:

```
GetScene()->UseSceneLight(false);  
  
mInfiniteLight = new InfiniteLight(0);  
mInfiniteLight->SetAzimuthElevation( 90.0f, 45.0f );  
mInfiniteLight->SetDiffuse( 255.0f, 255.0f, 255.0f, 1.0f );  
mInfiniteLight->SetSpecular( 255.0f, 255.0f, 255.0f, 1.0f);  
mInfiniteLight->SetEnabled( true );  
  
AddDrawable( mInfiniteLight.get() );
```

Many of the Delta3D test applications already do this.

Here is the API for the Config function call:

```
void Config (  
    dtCore::RefPtr<dtCore::InfiniteLight> il,  
    // a pointer to the scene's light source  
    dtCore::RefPtr<dtCore::Scene> scn,  
    // a pointer to the scene  
    osg::ref_ptr<Producer::Camera> ec,  
    // a pointer to the main camera from application scope:  
    // GetCamera()->GetCamera()  
    int texture_unit,  
    // the texture unit to be used  
    bool useUNC,  
    // flag determining if LSPSM shadows are used or Standard  
    // Orthogonal
```

```

int                                     sms,
// shadow map size
int                                     pof,
// Polygon Offset Facotr
int                                     pou
// Polygon Offset Units
);

```

This API defaults to a very generic Vertex and Pixel shader. If you want to use your own shader, you will have to write it such that it performs the correct texture coordinate transformations in the vertex program, and the correct texture lookup in the fragment program. You can follow my shaders in this case.

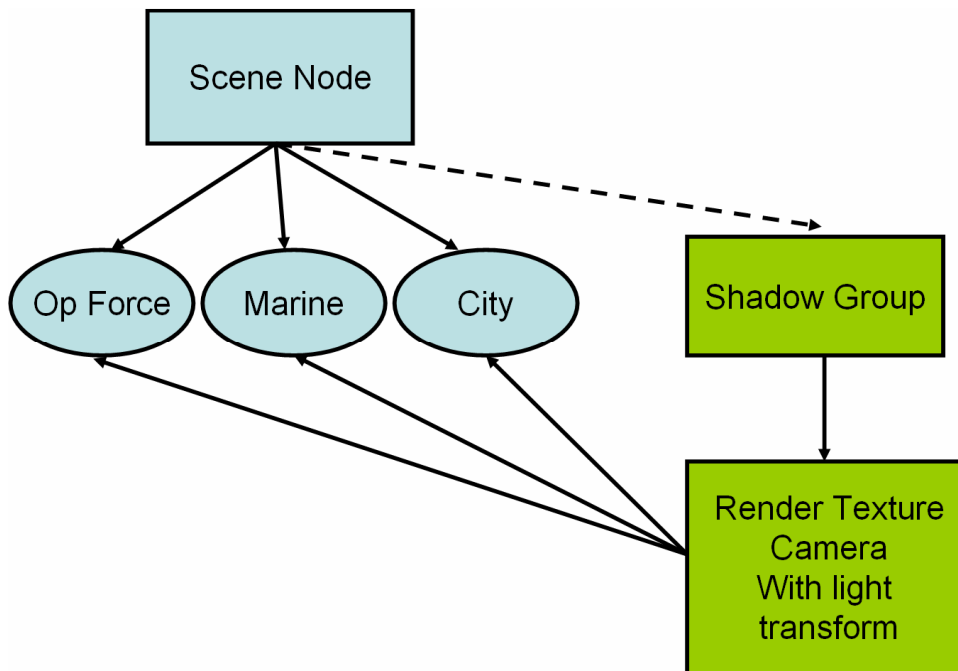


Figure 1: A shadowed group is added to the scene graph. The Render Texture Camera points to the drawable objects in the scenegraph. The Shadow Group is added and removed only for rendering so that this node does not interfere with the rest of Delta 3D.

Here is a basic idea of how the code works:

1. The library creates a shadow group and add:
 - a. An `osg::CameraNode` with a depth buffer and the `PRE_RENDER` order.
 - b. A pointer to the drawable object in the scene.
2. I create 2 call backs.

- a. The first is the actual shadow wrapper class it adds the Shadow Group to the Scene, and compute the shader parameters necessary for the shadows.
- b. The second call back removes the Shadow Group from the scene so that the Delta3D game engine does not traverse it and cause slowdowns, and collision problems.

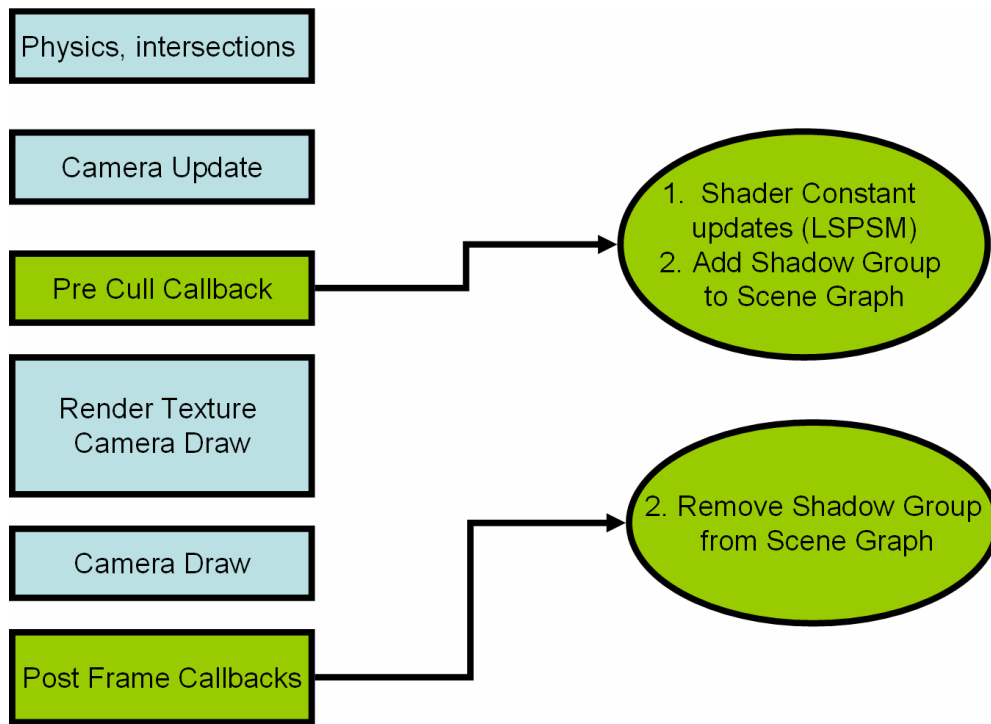


Figure 2: I add two call backs to the OSG Scene graph. The first happens directly after the main OSG camera is updated. This call back adds the shadowed scene to the scene graph, and updates the constants needed for shadows. The second removes the shadowed scene from the scene graph

Gotchas:

1. When objects are added to the scene graph a pointer must also be added to the Shadowed Group.
2. Shadows are very tightly coupled with the rest of the engine.
 - a. The same Uniform variable names must be used I set the following uniforms the vertex shader must use them correctly.

- i. uniform mat4 textureMat is the texture transform matrix
 - ii. uniform mat4 viewMatrixEyeI is the inverse view matrix for the eye.
 - b. Vertex shaders must transform the texture coordinates correctly
 - i. `vec4 world = viewMatrixEyeI* ecPos ;`
`gl_TexCoord[1] = textureMat * world ;`
 - c. Fragment shaders must perform the shadow lookup
 - i. `vec4 temp = shadow2DProj(shadowTexture,`
`gl_TexCoord[1]);`
 - ii. `gl_FragColor.xyz = colorlight.xyz * temp.xyz;`
3. You must also be sure not to over ride the Depth texture sampler with your own textures.



Figure 3: Results.