

# Making a Spectator

## Recommended Reading

[Introduction to Models](#)

## Video

[http://www.leadwerks.com/files/CPP/Making\\_A\\_Spectator.wmv](http://www.leadwerks.com/files/CPP/Making_A_Spectator.wmv)

## Required Files

[http://developer.leadwerks.com/Tutorials/CPP/Making\\_A\\_Spectator\\_Files.zip](http://developer.leadwerks.com/Tutorials/CPP/Making_A_Spectator_Files.zip)

## Spectator Mode

Many multiplayer games allow the player to view the action passively, as a free-flying non-interacting spectator. Spectator mode usually enables collision between the scene and the camera, although the spectator should not be able to interact with or affect any aspect of the game.

## Enabling Physics

We're going to use the physics system to make our spectator. We start with the code from the "Introduction to Models" lesson:

```
#include "engine.h"

inline float rnd( float min=0.0, float max=1.0 ) {
    return min + ((float)rand()/RAND_MAX)*(max-min);
}

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

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

    RegisterAbstractPath("C:\\Leadwerks Engine SDK");

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

    //Create a camera
    TEntity cam=CreateCamera();
    CameraClearColor(cam,Vec4(0,0,1,1));
    PositionEntity(cam,Vec3(0,2,-10));

    //Create a light
    TLight light=CreateDirectionalLight();
    RotateEntity(light,Vec3(45,45,0));

    //Create a render buffer
    TBuffer buffer=CreateBuffer(800,600,BUFFER_COLOR|BUFFER_DEPTH|BUFFER_NORMAL);

    Collisions(1,1,1);

    TModel scene=LoadModel("scene.gmf");
    if (!scene) {
        MessageBoxA(0,"Error","Failed to load mesh.",0);
        goto exitapp;
    }
    EntityType(scene,1);

    TModel model=LoadModel("oildrum.gmf");
    if (!model) {
        MessageBoxA(0,"Error","Failed to load mesh.",0);
        goto exitapp;
    }
    EntityType(model,1);
    SetBodyMass(model,1);

    for ( int n=1; n<=100; n++ ) {
        model=CopyEntity(model);
        PositionEntity(model,Vec3(rnd(-5,5),rnd(5,15),rnd(-5,5)));
        RotateEntity(model,Vec3(rnd(0,360),rnd(0,360),rnd(0,360)));
    }
}
```

```

TVec3 camrotation=Vec3(0);
float mx=0;
float my=0;
float move=0;
float strafe=0;

HideMouse();
MoveMouse(GraphicsWidth()/2,GraphicsHeight()/2);

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

    DebugPhysics(KeyDown(KEY_P));

    //Camera look
    mx=Curve(MouseX()-GraphicsWidth()/2,mx,6);
    my=Curve(MouseY()-GraphicsHeight()/2,my,6);
    MoveMouse(GraphicsWidth()/2,GraphicsHeight()/2);

    camrotation.X=camrotation.X+my/10.0;
    camrotation.Y=camrotation.Y-mx/10.0;
    RotateEntity(cam,camrotation);

    //Camera movement
    move=Curve(KeyDown(KEY_W)-KeyDown(KEY_S),move,20);
    strafe=Curve(KeyDown(KEY_D)-KeyDown(KEY_A),strafe,20);
    MoveEntity(cam,Vec3(strafe/10.0,0,move/10.0));

    //Update the world
    UpdateWorld();

    //Render the scene
    SetBuffer(buffer);
    RenderWorld();

    //Render lighting
    SetBuffer(BackBuffer());
    RenderLights(buffer);

    //Swap the front and back buffer
    Flip();
}

exitapp:
return Terminate();
}

```

The above code allows us to fly around and look at the scene, but because it is not physics-based, there is no camera collision.

First let's add a physics body for the spectator. Add this code before the main loop:

```
TBody spectator=CreateBodySphere();
```

We want the spectator to be a dynamic body, so let's set the mass to 1.0:

```
SetBodyMass(spectator,1);
```

The spectator should float in the air. To disable gravity for the spectator body, use the SetBodyGravityMode command:

```
SetBodyGravityMode(spectator,0);
```

To make the camera follow the spectator body, add this code in the main loop after the call to UpdateWorld and before the call to RenderWorld:

```
PositionEntity(cam,EntityPosition(spectator));
```

Here is our complete code:

```
#include "engine.h"

inline float rnd( float min=0.0, float max=1.0 ) {
    return min + ((float)rand()/RAND_MAX)*(max-min);
}

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

    RegisterAbstractPath("C:\\Leadwerks Engine SDK");

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

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

    //Create a camera
    TEntity cam=CreateCamera();
    CameraClearColor(cam,Vec4(0,0,1,1));
    PositionEntity(cam,Vec3(0,2,-10));

    //Create a light
    TLight light=CreateDirectionalLight();
    RotateEntity(light,Vec3(45,45,0));
}
```

```

//Create a render buffer
TBuffer buffer=CreateBuffer(800,600,BUFFER_COLOR|BUFFER_DEPTH|BUFFER_NORMAL);

Collisions(1,1,1);

TModel scene=LoadModel("scene.gmf");
if (!scene) {
    MessageBoxA(0,"Error","Failed to load mesh.",0);
    goto exitapp;
}
EntityType(scene,1);

TModel model=LoadModel("oildrum.gmf");
if (!model) {
    MessageBoxA(0,"Error","Failed to load mesh.",0);
    goto exitapp;
}
EntityType(model,1);
SetBodyMass(model,1);

for ( int n=1; n<=100; n++ ) {
    model=CopyEntity(model);
    PositionEntity(model,Vec3(rnd(-5,5),rnd(5,15),rnd(-5,5)));
    RotateEntity(model,Vec3(rnd(0,360),rnd(0,360),rnd(0,360)));
}

//Create the spectator
TBody spectator=CreateBodySphere();
SetBodyMass(spectator,1);
SetBodyGravityMode(spectator,0);

TVec3 camrotation=Vec3(0);
float mx=0;
float my=0;
float move=0;
float strafe=0;

HideMouse();
MoveMouse(GraphicsWidth()/2,GraphicsHeight()/2);

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

    DebugPhysics(KeyDown(KEY_P));

    //Camera look
    mx=Curve(MouseX()-GraphicsWidth()/2,mx,6);
    my=Curve(MouseY()-GraphicsHeight()/2,my,6);
    MoveMouse(GraphicsWidth()/2,GraphicsHeight()/2);

    camrotation.X=camrotation.X+my/10.0;
    camrotation.Y=camrotation.Y-mx/10.0;
    RotateEntity(cam,camrotation);

    //Camera movement
    move=Curve(KeyDown(KEY_W)-KeyDown(KEY_S),move,20);
    strafe=Curve(KeyDown(KEY_D)-KeyDown(KEY_A),strafe,20);
    MoveEntity(cam,Vec3(strafe/10.0,0,move/10.0));

    //Update the world
    UpdateWorld();
}

```

```

    //Position the camera
    PositionEntity(cam,EntityPosition(spectator));

    //Render the scene
    SetBuffer(buffer);
    RenderWorld();

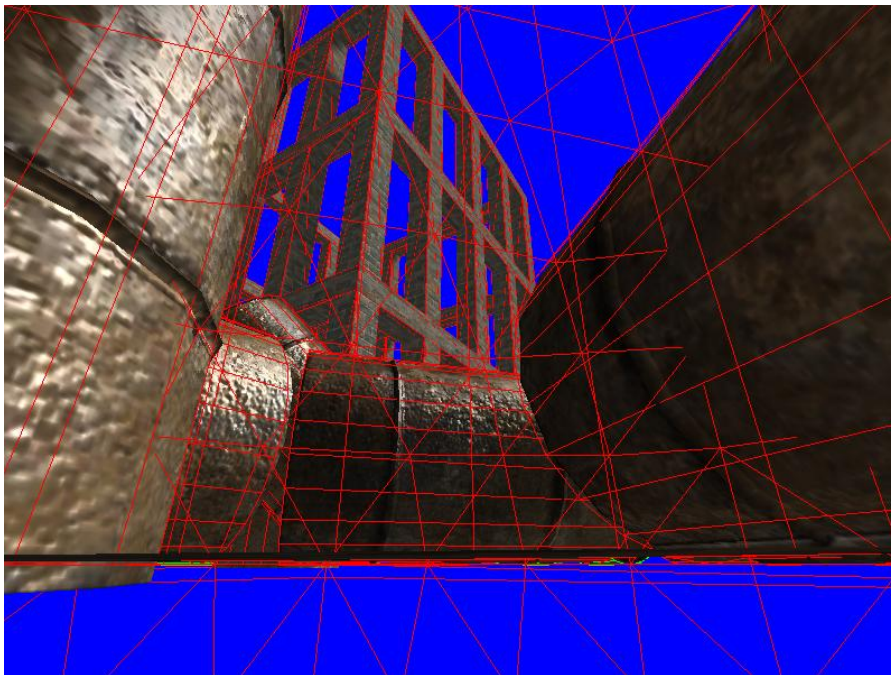
    //Render lighting
    SetBuffer(BackBuffer());
    RenderLights(buffer);

    //Swap the front and back buffer
    Flip();
}

exitapp:
return Terminate();
}

```

If we run this we will see the camera is stuck in the middle of the scene. Though we can look around, we can't move at all. This is because the camera is being repositioned each frame before `RenderWorld` is called. If we press the P key to enable physics debugging, we can see the camera is stuck inside the spectator sphere body:



### Adding Movement

Now let's alter the camera controls. Find the camera movement code in the main loop and comment it out:

```
//Camera movement
//move=Curve(KeyDown(KEY_W)-KeyDown(KEY_S),move,20);
//strafe=Curve(KeyDown(KEY_D)-KeyDown(KEY_A),strafe,20);
//MoveEntity(cam,Vec3(strafe/10.0,0,move/10.0));
```

Below this section add the following code. The Curve function is no longer necessary because we are going to let the physics system handle acceleration and inertia:

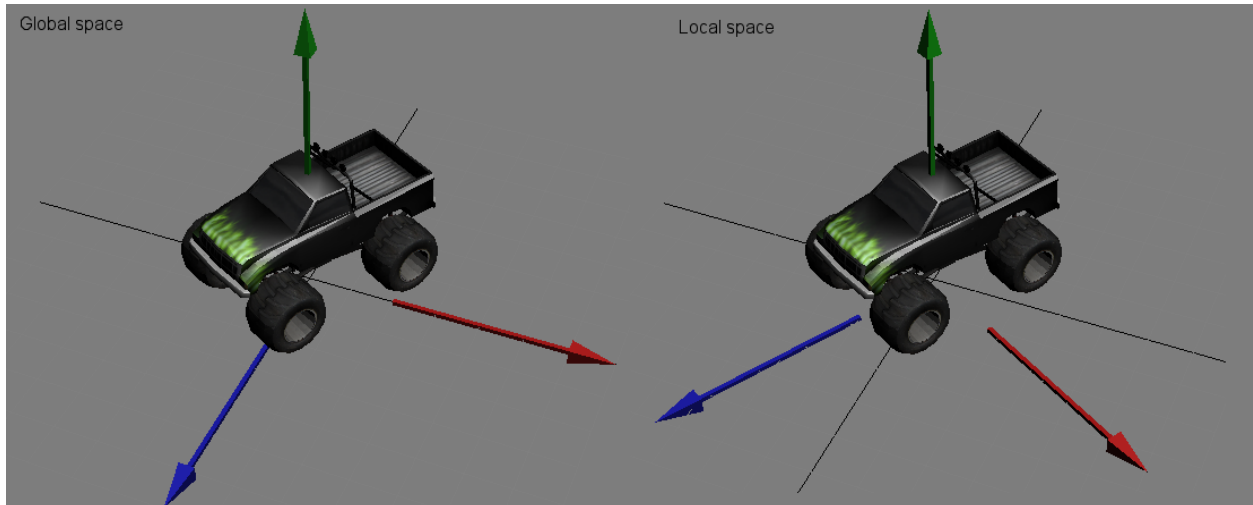
```
move=KeyDown(KEY_W)-KeyDown(KEY_S);
strafe=KeyDown(KEY_D)-KeyDown(KEY_A);
```

Commands like MoveEntity, PositionEntity, RotateEntity, and other commands that change an entity's orientation should never be used on bodies. These will reset the body matrix, causing the physics simulation to reset. Instead of moving the spectator body like we did the camera, we are going to use forces to move it. The AddBodyForce command will exert a force in local or global space. Local space means relative to the body's own orientation, and global space means relative to the world. (By default, force is added in global space.)

How do we find the force to apply? We can try adding the force using the move and strafe values from above:

```
TVec3 force = Vec3(strafe,0,move);
AddBodyForce(spectator,force);
```

If you run the program at this point, you will see that this does not make the spectator move in the correct directions. Instead we want to move relative to the camera's rotation. The image below illustrates this idea of relative vectors:



Global space versus local space. On the left, global space is shown to be relative to the world axes.

On the right, local space is shown to be relative to the truck model.

To transform a vector from one entity's space to another, we use the `TFormVector` command. This function accepts a vector, a source entity, and a destination entity. If either the source or destination entities are set to 0, the engine will use the world space. We want to transform the movement vector from the camera space to world space, so we will do the following:

```
TVec3 force = Vec3(strafe,0,move);
force=TFormVector(force,cam,0);
```

This will give us a usable vector we can then add to the body as a force:

```
AddBodyForce(spectator,force);
```

Here is our final camera control code:

```
//Camera movement
//move=Curve(KeyDown(KEY_W)-KeyDown(KEY_S),move,20);
//strafe=Curve(KeyDown(KEY_D)-KeyDown(KEY_A),strafe,20);
//MoveEntity(cam,Vec3(strafe/10.0,0,move/10.0));
move=KeyDown(KEY_W)-KeyDown(KEY_S);
strafe=KeyDown(KEY_D)-KeyDown(KEY_A);
TVec3 force = Vec3(strafe,0,move);
force=TFormVector(force,cam,0);
AddBodyForce(spectator,force);
```

## Fine Tuning

At this point, our spectator moves very smoothly, but it feels “sloppy”. It is hard to change directions, and when we release the keys it takes a long time to come to a stop. We can use damping to eliminate this problem. Add this code after the spectator is created and before the main loop starts:

```
SetBodyDamping(spectator,1.0);
```

This will reduce the spectator body’s velocity each physics step. Because we are damping the velocity, we need to increase the force we add to the body. Find this line in your source code:

```
TVec3 force = Vec3(strafe,0,move);
```

Change it to this code, which will add ten times as much force:

```
TVec3 force = Vec3(strafe*10.0,0,move*10.0);
```

Here is what your code should look like at this point. If you run it, you will see that the spectator controls now feel very responsive and smooth:

```
#include "engine.h"

inline float rnd( float min=0.0, float max=1.0 ) {
    return min + ((float)rand()/RAND_MAX)*(max-min);
}

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

    RegisterAbstractPath("C:\\Leadwerks Engine SDK");

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

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

    //Create a camera
    TEntity cam=CreateCamera();
    CameraClearColor(cam,Vec4(0,0,1,1));
    PositionEntity(cam,Vec3(0,2,-10));

    //Create a light
```

```

TLight light=CreateDirectionalLight();
RotateEntity(light,Vec3(45,45,0));

//Create a render buffer
TBuffer buffer=CreateBuffer(800,600,BUFFER_COLOR|BUFFER_DEPTH|BUFFER_NORMAL);

Collisions(1,1,1);

TModel scene=LoadModel("scene.gmf");
if (!scene) {
    MessageBoxA(0,"Error","Failed to load mesh.",0);
    goto exitapp;
}
EntityType(scene,1);

TModel model=LoadModel("oildrum.gmf");
if (!model) {
    MessageBoxA(0,"Error","Failed to load mesh.",0);
    goto exitapp;
}
EntityType(model,1);
SetBodyMass(model,1);

for ( int n=1; n<=100; n++ ) {
    model=CopyEntity(model);
    PositionEntity(model,Vec3(rnd(-5,5),rnd(5,15),rnd(-5,5)));
    RotateEntity(model,Vec3(rnd(0,360),rnd(0,360),rnd(0,360)));
}

//Create the spectator
TBody spectator=CreateBodySphere();
SetBodyMass(spectator,1);
SetBodyGravityMode(spectator,0);
SetBodyDamping(spectator,1.0);

TVec3 camrotation=Vec3(0);
float mx=0;
float my=0;
float move=0;
float strafe=0;

HideMouse();
MoveMouse(GraphicsWidth()/2,GraphicsHeight()/2);

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

    DebugPhysics(KeyDown(KEY_P));

    //Camera look
    mx=Curve(MouseX()-GraphicsWidth()/2,mx,6);
    my=Curve(MouseY()-GraphicsHeight()/2,my,6);
    MoveMouse(GraphicsWidth()/2,GraphicsHeight()/2);

    camrotation.X=camrotation.X+my/10.0;
    camrotation.Y=camrotation.Y-mx/10.0;
    RotateEntity(cam,camrotation);

    //Camera movement
    //move=Curve(KeyDown(KEY_W)-KeyDown(KEY_S),move,20);
    //strafe=Curve(KeyDown(KEY_D)-KeyDown(KEY_A),strafe,20);
    //MoveEntity(cam,Vec3(strafe/10.0,0,move/10.0));
    move=KeyDown(KEY_W)-KeyDown(KEY_S);

```

```

    strafe=KeyDown(KEY_D)-KeyDown(KEY_A);
    TVec3 force = Vec3(strafe*10.0,0,move*10.0);
    force=TFormVector(force,cam,0);
    AddBodyForce(spectator,force);

    //Update the world
    UpdateWorld();

    //Position the camera
    PositionEntity(cam,EntityPosition(spectator));

    //Render the scene
    SetBuffer(buffer);
    RenderWorld();

    //Render lighting
    SetBuffer(BackBuffer());
    RenderLights(buffer);

    //Swap the front and back buffer
    Flip();
}

exitapp:
return Terminate();
}

```

### Collision

Collision between collision types one and one is already enabled, so to add collision between the spectator and the scene, we only have to set the entity type:

```

EntityType(spectator,1);

```

However, this will cause some unwanted behavior. If you try the code above, the oil drums will crash down on the spectator, knocking the camera around. The spectator will be able to push over the oil drums. When describing characteristics of a spectator, we said that it should not interfere with the game. Unless you are making a poltergeist simulator, it would be a strange experience for players who see objects knocked over by an invisible spectator. To solve this problem we will make more advanced use of collision types.

First we need to decide what types of objects we have, categorized by collision. In this program we have the static scene model, the spectator body, and the oil drum models. Let's make a chart to show how they should interact. "Yes" means the object types should collide, and "No" means they should not.

	Scene	Oil drum	Spectator
Scene	No	Yes	Yes
Oil drum	Yes	Yes	No
Spectator	Yes	No	No

From the chart above, we can see the scene should collide with the oil drums and the spectator. The spectator should collide with only the scene. The oil drums should collide with the scene, as well as with other oil drums.

Let's assign a collision type for each of these groups. We will use 1 for the scene, 2 for the oil drums, and 3 for the spectator. Our collision responses should then be as follows:

```
Collisions (1, 1, 0);  
Collisions (1, 2, 1);  
Collisions (1, 3, 1);  
Collisions (2, 1, 1);  
Collisions (2, 2, 1);  
Collisions (2, 3, 0);  
Collisions (3, 1, 1);  
Collisions (3, 2, 0);  
Collisions (3, 3, 0);
```

It is not necessary to specify if no collision response should occur, unless we are changing a previously defined response. Therefore we can remove all the collision definitions that define no response should occur:

```
Collisions (1, 2, 1);  
Collisions (1, 3, 1);  
Collisions (2, 1, 1);  
Collisions (2, 2, 1);  
Collisions (3, 1, 1);
```

We can also remove redundant collision definitions as they are unnecessary. For example, if the response for collisions between type 1 and 2 has been defined, it is not necessary to define a response for the same types in opposite order. Find the line where we previously defined a collision response between type 1 and 1, and replace it with this:

```
Collisions (1, 2, 1);  
Collisions (1, 3, 1);  
Collisions (2, 2, 1);
```

Find the line where we set the first oil drum model's collision type, and change it to 2. All copies of the model will inherit the same collision type.

Find the line where we set the spectator body's collision type, and change it to 3.

Finally, let's reposition the spectator so that it starts at a better position. Add this code after the spectator has been created, and before the main loop:

```
PositionEntity(spectator,Vec3(0,2,-10));
```

Here is our final code. The spectator will collide with the static scene, but won't collide with the dynamic bodies. This allows the player to view a game without disturbing the action:

```
#include "engine.h"

inline float rnd( float min=0.0, float max=1.0 ) {
    return min + ((float)rand()/RAND_MAX)*(max-min);
}

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

    RegisterAbstractPath("C:\\Leadwerks Engine SDK");

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

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

    //Create a camera
    TEntity cam=CreateCamera();
    CameraClearColor(cam,Vec4(0,0,1,1));
    PositionEntity(cam,Vec3(0,2,-10));

    //Create a light
    TLight light=CreateDirectionalLight();
    RotateEntity(light,Vec3(45,45,0));

    Collisions(1,2,1);
    Collisions(1,3,1);
    Collisions(2,2,1);

    //Create a render buffer
    TBuffer buffer=CreateBuffer(800,600,BUFFER_COLOR|BUFFER_DEPTH|BUFFER_NORMAL);

    Collisions(1,1,1);

    TModel scene=LoadModel("scene.gmf");
    if (!scene) {
        MessageBoxA(0,"Error","Failed to load mesh.",0);
        goto exitapp;
    }
    EntityType(scene,1);

    TModel model=LoadModel("oildrum.gmf");
    if (!model) {
        MessageBoxA(0,"Error","Failed to load mesh.",0);
        goto exitapp;
    }
    EntityType(model,2);
}
```

```

SetBodyMass(model,1);

for ( int n=1; n<=100; n++ ) {
    model=CopyEntity(model);
    PositionEntity(model,Vec3(rnd(-5,5),rnd(5,15),rnd(-5,5)));
    RotateEntity(model,Vec3(rnd(0,360),rnd(0,360),rnd(0,360)));
}

//Create the spectator
TBody spectator=CreateBodySphere();
SetBodyMass(spectator,1);
SetBodyGravityMode(spectator,0);
SetBodyDamping(spectator,1.0);
EntityType(spectator,3);
PositionEntity(spectator,Vec3(0,2,-10));

TVec3 camrotation=Vec3(0);
float mx=0;
float my=0;
float move=0;
float strafe=0;

HideMouse();
MoveMouse(GraphicsWidth()/2,GraphicsHeight()/2);

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

    DebugPhysics(KeyDown(KEY_P));

    //Camera look
    mx=Curve(MouseX()-GraphicsWidth()/2,mx,6);
    my=Curve(MouseY()-GraphicsHeight()/2,my,6);
    MoveMouse(GraphicsWidth()/2,GraphicsHeight()/2);

    camrotation.X=camrotation.X+my/10.0;
    camrotation.Y=camrotation.Y-mx/10.0;
    RotateEntity(cam,camrotation);

    //Camera movement
    //move=Curve(KeyDown(KEY_W)-KeyDown(KEY_S),move,20);
    //strafe=Curve(KeyDown(KEY_D)-KeyDown(KEY_A),strafe,20);
    //MoveEntity(cam,Vec3(strafe/10.0,0,move/10.0));
    move=KeyDown(KEY_W)-KeyDown(KEY_S);
    strafe=KeyDown(KEY_D)-KeyDown(KEY_A);
    TVec3 force = Vec3(strafe*10.0,0,move*10.0);
    force=TFormVector(force,cam,0);
    AddBodyForce(spectator,force);

    //Update the world
    UpdateWorld();

    //Position the camera
    PositionEntity(cam,EntityPosition(spectator));

    //Render the scene
    SetBuffer(buffer);
    RenderWorld();

    //Render lighting
    SetBuffer(BackBuffer());
    RenderLights(buffer);
}

```

```
        //Swap the front and back buffer
        Flip();
    }

    exitapp:
    return Terminate();
}
```

Copyright Leadwerks Software 2008

All Rights Reserved.