

Character Controllers

Recommended Reading

[Making a Spectator](#)

Video

http://www.leadwerks.com/files/Tutorials/CPP/Character_Controllers.wmv

Required Files

http://developer.leadwerks.com/Tutorials/CPP/Character_Controllers_Files.zip

Character Controllers

Character controllers are an important yet often overlooked feature for any 3D development package. Movement controls are the most immediate form of interaction a player has with the virtual environment. Any glitches in these controls will create an unpleasant experience for the end user. Fortunately, Leadwerks Engine provides a robust and stable character controller that interacts seamlessly with other physical objects. This controller can be used for any character, whether the input comes from the keyboard, AI, or a remote player in a networked game.

When setting out to design our character controller, we had a few requirements in mind:

- Controller should not slide down slopes below a maximum slope angle.
- Controllers should climb obstacles below the maximum step height. Climbing should be instant, with no loss of horizontal velocity, or “struggling” to climb stairs.
- Controllers should be able to interact with other physics bodies. Controllers should be able to push and be pushed by physics bodies.
- Jitters and other glitches were absolutely unacceptable; player control must feel smooth and fluid.

These behaviors were very difficult to implement, but after spending many months on the problem we achieved excellent results. In this lesson we will learn to make a controller with the above features, as well as running and jumping behavior.

Getting Started

We start with this simple program, which loads a model and renders it. Lighting is provided by a single point light placed in the center of the scene:

```
#include "engine.h"

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

    //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));

    //Load a model
    TModel scene=LoadModel("scene.gmf");
    if (scene==0) {
        MessageBoxA(0,"Error","Failed to load model.",0);
        goto exitapp;
    }

    //Create a light
    TLight light=CreatePointLight();
    PositionEntity(light,Vec3(0,5,0));

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

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

        //Update the world
        UpdateWorld();

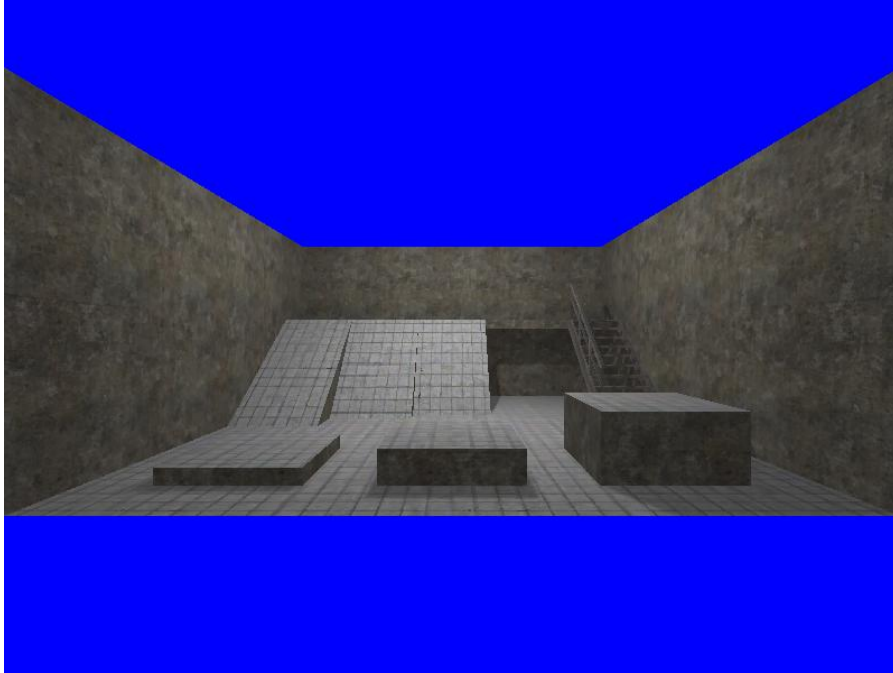
        //Render the scene
        SetBuffer(buffer);
        RenderWorld();

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

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

    exitapp:
    return Terminate();
}
```

Here is the result:



First, let's set up collisions. Add this code before the main loop, after the scene model is loaded:

```
EntityType(scene,2);  
Collisions(1,2,true);
```

Now we create a character controller with the CreateController command. Add this code before the main loop:

```
TController player=CreateController();  
EntityType(player,1);
```

Now we need to declare a few variable for movement. Add this code before the main loop:

```
float move=0.0;  
float strafe=0.0;
```

Let's turn physics debugging on so we can see the controller. Add this code before the main loop:

```
DebugPhysics(1);
```

Now we are going to get the keyboard input and use it for the movement values. Add this code inside the main loop before the call to UpdateWorld:

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

Finally, we apply the movement to the controller using the UpdateController command:

```
UpdateController(player,0.0,move,strafe);
```

Here is our finished code:

```
#include "engine.h"  
  
int main(int argc, char** argv)  
{  
    Initialize();  
  
    //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));  
  
    //Load a model  
    TModel scene=LoadModel("scene.gmf");  
    if (scene==0) {  
        MessageBoxA(0,"Error","Failed to load model.",0);  
        goto exitapp;  
    }  
    EntityType(scene,2);  
  
    Collisions(1,2,true);  
  
    //Create a light  
    TLight light=CreatePointLight();
```

```

PositionEntity(light,Vec3(0,5,0));

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

TController player=CreateController();
EntityType(player,1);

float move=0.0;
float strafe=0.0;

DebugPhysics(1);

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

    //Player movement
    move=KeyDown(KEY_W)-KeyDown(KEY_S);
    strafe=KeyDown(KEY_D)-KeyDown(KEY_A);

    //Set controller input
    UpdateController(player,0.0,move,strafe);

    //Update the world
    UpdateWorld();

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

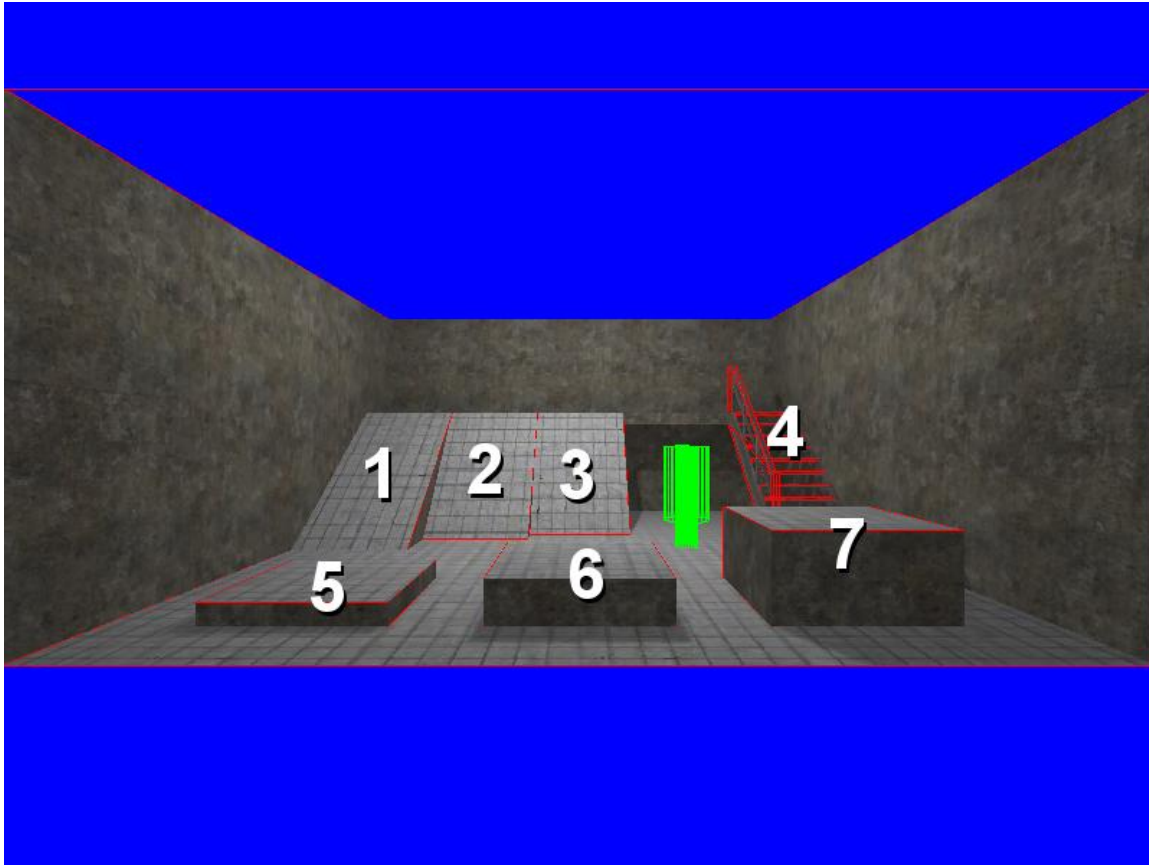
    //Render lighting
    SetBuffer(BackBuffer());
    RenderLights(buffer);

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

exitapp:
return Terminate();
}

```

When we run the program, we will see a visible controller we can move using the WASD keys.



Take note of the following:

1. The controller can climb this slope because it is below the maximum slope angle.
2. The controller can climb this slope because it equal to the maximum slope angle.
3. The controller cannot climb this slope because it is steeper than the maximum slope angle.
4. The controller can climb the stairs effortlessly.
5. The controller can step up this block because it is below the maximum step height.
6. The controller can step up this block because it is equal to the maximum step height.
7. The controller cannot step up this block because it is above the maximum step height.

First Person View

We can position the camera to make first person player controls.

Comment out this line in your code:

```
DebugPhysics(1);
```

Declare these variables before the main loop:

```
TVec3 camrotation=Vec3(0);  
float mx=0;  
float my=0;
```

Now center the mouse and hide the cursor. Add this code right before the main loop:

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

Now we set up camera looking, just like we did in the “Making a Spectator” lesson. Add this code in the main loop, at the beginning:

```
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);
```

Now we will pass the Y component of the camera rotation to the UpdateController function. Find this line in your code:

```
UpdateController(player,0.0,move,strafe);
```

Replace it with this:

```
UpdateController(player,camrotation.Y,move,strafe);
```

Finally, we position the camera. The character controller’s position is at the very center and bottom of the physics body, so we will use the controller position plus the eye height. The eye height should be a little less than the controller height:

```
TVec3 playerpos=EntityPosition(player);  
playerpos.Y+=1.75;  
PositionEntity(cam,playerpos);
```

Here is our complete code which provides first person player controls:

```
#include "engine.h"

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

    //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));

    //Load a model
    TModel scene=LoadModel("scene.gmf");
    if (scene==0) {
        MessageBoxA(0,"Error","Failed to load model.",0);
        goto exitapp;
    }
    EntityType(scene,2);

    Collisions(1,2,true);

    //Create a light
    TLight light=CreatePointLight();
    PositionEntity(light,Vec3(0,5,0));

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

    TController player=CreateController();
    EntityType(player,1);

    float move=0.0;
    float strafe=0.0;
    TVec3 camrotation=Vec3(0);
    float mx=0;
    float my=0;

    //DebugPhysics(1);

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

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

        //Camera looking
        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);

        //Player movement
        move=KeyDown(KEY_W)-KeyDown(KEY_S);
        strafe=KeyDown(KEY_D)-KeyDown(KEY_A);

        //Set controller input
        UpdateController(player,camrotation.Y,move,strafe);

        //Update the world
        UpdateWorld();

        //Position the camera
        TVec3 playerpos=EntityPosition(player);
        playerpos.Y+=1.75;
        PositionEntity(cam,playerpos);

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

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

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

    exitapp:
    return Terminate();
}

```

Fine Tuning

Game physics tend to look better with a gravity value higher than the default value of -9.8 meters per second squared. This helps eliminate the “floaty” feeling that real-time physics sometimes have. Add this code before the main loop:

```
SetWorldGravity(Vec3(0,-20,0));
```

We can also make stair climbing feel more natural by using a small smoothing adjustment for the camera height. Find this block of code in your source:

```

TVec3 playerpos=EntityPosition(player);
playerpos.Y+=1.75;
PositionEntity(cam,playerpos);

```

Replace it with this:

```
TVec3 playerpos=EntityPosition(player);
TVec3 camerapos=EntityPosition(cam);
camerapos.Y=Curve(playerpos.Y+1.75, camerapos.Y, 2.0);
camerapos=Vec3(playerpos.X, camerapos.Y, playerpos.Z);
PositionEntity(cam, camerapos);
```

Run the program now to see these improvements.

Jump Around

We can add controls to make the player jump. Add this code in the main loop, before the call to UpdateController. The ControllerAirborne function will tell us if the controller is on the ground or not.

```
float jump=0.0;
if (KeyHit(KEY_SPACE)) {
    if (!ControllerAirborne(player)) {
        jump=4.0;
    }
}
```

Now add the jump parameter at the end of the call to UpdateController:

```
UpdateController(player, camrotation.Y, move, strafe, jump);
```

You can now make the player jump by pressing the space key.

Run Player, Run

We can add running behavior by increasing the move and strafe values when the left or right shift keys are pressed. Add this code in the main loop, after the move and strafe values are set and before the call to UpdateController:

```
if (KeyDown(KEY_LSHIFT) || KeyDown(KEY_RSHIFT)) {
    if (!ControllerAirborne(player)) {
        move*=2.0;
        strafe*=2.0;
    }
}
```

Here is our finished controller code. This will provide smooth and capable player movement. Experiment with the movement values until you find the settings you like. The UpdateController command also has additional optional values you can adjust:

```
#include "engine.h"

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

    //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));

    Collisions(1,2,true);

    //Load a model
    TModel scene=LoadModel("scene.gmf");
    if (scene==0) {
        MessageBoxA(0,"Error","Failed to load model.",0);
        goto exitapp;
    }
    EntityType(scene,2);

    //Create a light
    TLight light=CreatePointLight();
    PositionEntity(light,Vec3(0,5,0));

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

    TController player=CreateController();
    EntityType(player,1);
    SetBodyDamping(player,0.0);
    SetWorldGravity(Vec3(0,-20,0));

    float move=0.0;
    float strafe=0.0;
    TVec3 camrotation=Vec3(0);
    float mx=0;
    float my=0;

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

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

        //Camera looking
        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);

//Player movement
move=KeyDown(KEY_W)-KeyDown(KEY_S);
strafe=KeyDown(KEY_D)-KeyDown(KEY_A);

//Jumping
float jump=0.0;
if (KeyHit(KEY_SPACE)) {
    if (!ControllerAirborne(player)) {
        jump=4.0;
    }
}

//Run
if (KeyDown(KEY_LSHIFT)||KeyDown(KEY_RSHIFT)) {
    if (!ControllerAirborne(player)) {
        move*=2.0;
        strafe*=2.0;
    }
}

//Set controller input
UpdateController(player,camrotation.Y,move,strafe,jump);

//Update the world
UpdateWorld();

//Position the camera
TVec3 playerpos=EntityPosition(player);
TVec3 camerapos=EntityPosition(cam);
camerapos.Y=Curve(playerpos.Y+1.75,camerapos.Y,2.0);
camerapos=Vec3(playerpos.X,camerapos.Y,playerpos.Z);
PositionEntity(cam,camerapos);

//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.