

Introduction to Models

Recommended Reading

[Rendering Lights](#)

[Camera Controls](#)

[Introduction to Meshes](#)

[Introduction to Bodies](#)

Video

http://www.leadwerks.com/files/Tutorials/Cpp/Introduction_To_Models.wmv

Required Files

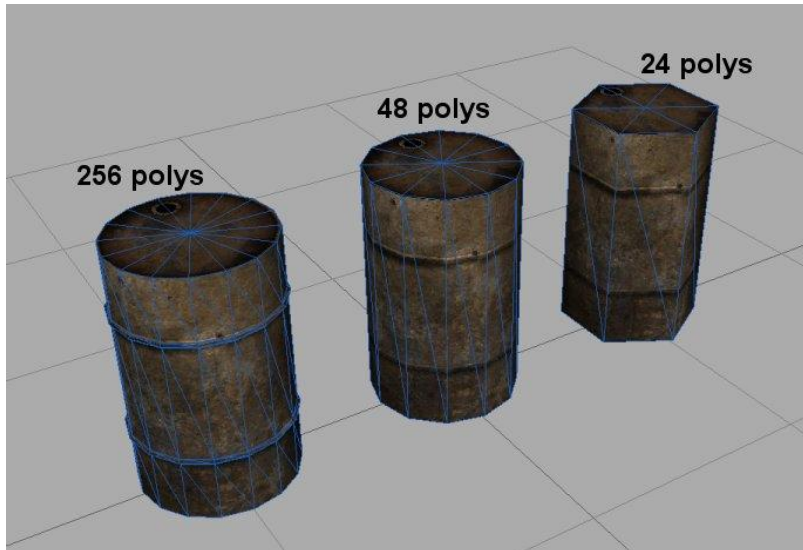
http://www.leadwerks.com/files/Tutorials/Cpp/Introduction_To_Models_Files.zip

All About Models

Models are an entity class that combines visible meshes with physics bodies. Models are an extension of the body class, so they are considered physical objects. A model itself is not visible, but a model has one or more children which are visible meshes. These child entities are hidden and shown depending on distance from the camera, to achieve LOD behavior; a model will appear more detailed when it is close and a less detailed version will be rendered when it is far away. The relationship between models and meshes is analogous to that between materials and textures; Meshes store the raw data and models contain additional info for displaying and managing them. The characteristics of models as both a physical and visual object make these the most commonly used type of object.

LOD

When objects are far away from the camera, small geometry details are not noticeable, yet they still consume processing power. A low resolution mesh can look just as good from a distance, and use less processing power. This low resolution mesh is called a level-of-detail (LOD) mesh.



A model's children are ordered by detail level:

- Model
 - LOD Mesh 1 (max detail)
 - LOD Mesh 2
 - LOD Mesh 3 (min detail)

When a model is rendered, an LOD mesh is chosen based on the model's distance from the camera. The current LOD mesh is shown, and the model's other LOD meshes are hidden.

Loading Models

A mesh can be loaded as a model in the same way a texture can be loaded as a material. The `LoadModel` command will accept any valid mesh file name as a parameter:

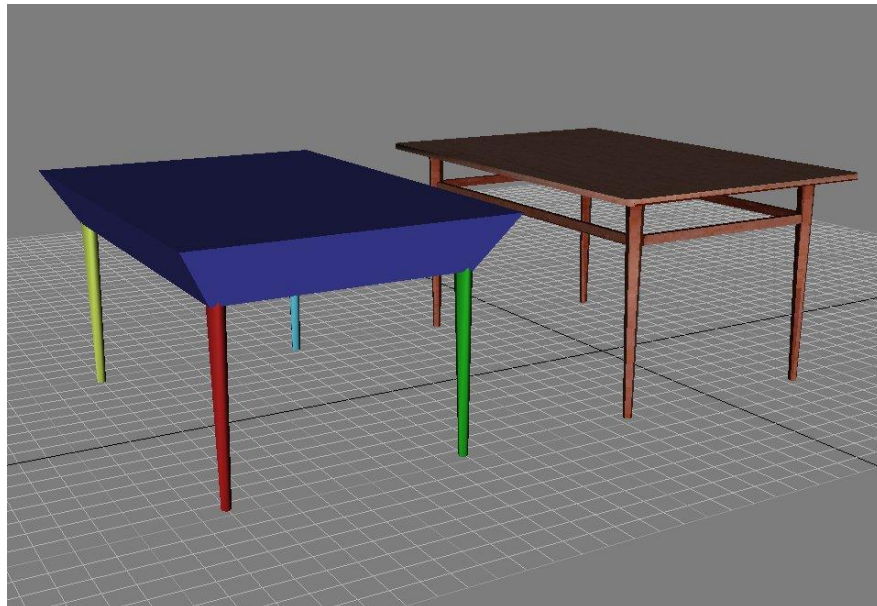
```
LoadModel("oildrum.gmf");
```

We can add LOD meshes by naming the files the same as the main mesh, with the suffix "LOD" and the detail level number. If the files "oildrumLOD1.gmf" is found in the same folder as "oildrum.gmf", it will be loaded and used as the second detail level. To add another detail level we change the "LOD1" suffix to "LOD2".

Model Physics

Since models are bodies, they must have a physical collision shape. By default the engine will generate a collision tree based on the geometry of the lowest detail level mesh. We can also specify a collision shape by creating a PHY collision file and placing it in the same folder the meshes are loaded from. If the file "oildrum.phy" is found in the same folder as "oildrum.gmf" it will be loaded and used for the collision shape.

Physics collision files can be created with the Phygen tool. They can take the shape of a box, cylinder, cone, sphere, capsule, chamfer cylinder, convex hull, or collision tree. Phygen can generate convex hulls and collision by loading an OBJ file. Convex hull collisions can be made up of multiple convex shapes in the OBJ file. Collision tree bodies are always static, even if the mass is greater than zero. Therefore dynamic bodies must always have a physics collision file present. Use primitives like boxes or spheres for basic shapes and convex hulls for more complex dynamic bodies.



Visual mesh on the right and physics collision shape on the left, made from convex hulls.

Pulling it All Together

Let's start with a program that sets up camera controls.

```
#include "engine.h"

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);

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)) {

    //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();  
}
```

Before doing anything, let's enable collisions between entities of type one and one. Add this code before the main loop:

```
Collisions(1,1,1);
```

Let's load the scene model and set the collision type. Add this code before the main loop:

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

Now let's load an oil drum mesh and set the collision type. Add this code before the main loop:

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

Because models are an extension of the body class, body commands work on models. Add this code after the previous block. This will set the mass to 1 and make a dynamic body:

```
SetBodyMass(model,1);
```

Just to make it more interesting, let's make a bunch of copies of the model. Because a PHY file is being loaded from the same folder as the meshes, these bodies will be dynamic. Add this code after the previous block:

```
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)));  
}
```

The `rnd` function is not built-in to the C language, so let's add this function at the top of the code, before the main function. This will return a random decimal value between the min and max values passed to the function:

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

Finally, let's add this line of code in the main loop to enable physics visualization when a key is pressed:

```
DebugPhysics(KeyDown(KEY_P));
```

Here is our final 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)));
    }

    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);

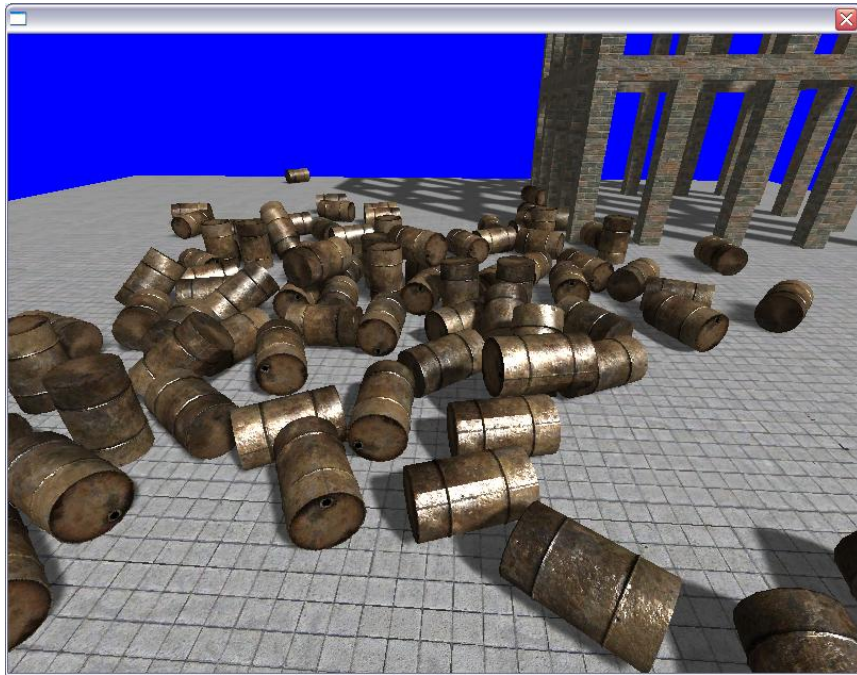
        //Swap the front and back buffer
        Flip();
    }

    exitapp:
    return Terminate();

```

}

With physics visualization enabled we can see both the mesh and the physics aspects of models. If you move the camera further away you may be able to see a transition when models switch to a lower detail level, but good artistry minimizes this phenomenon.



Copyright Leadwerks Software 2008

All Rights Reserved.