

# Collision and Raycasting

## Recommended Reading

[Rendering Lights](#)

[Introduction to Meshes](#)

[Introduction to Bodies](#)

## Video

[http://www.leadwerks.com/files/Tutorials/CPP/Collision\\_And\\_Raycasting.wmv](http://www.leadwerks.com/files/Tutorials/CPP/Collision_And_Raycasting.wmv)

## Introduction

Collision and raycasting are the primary techniques with which a programmer interacts with the physical world of a game. These techniques form the basis of most gameplay mechanics. Ballistics and artificial intelligence rely heavily on raycasting, while triggered sequences and other features require a robust collision system. This tutorial will establish a foundation upon which further lessons will demonstrate advanced gameplay mechanics.

## Retrieving Collision Results

Collision events can be evaluated as they occur using an entity collision callback. This allows us to handle collision events without permanently storing the data anywhere. When the entity collision callback is called, the colliding entities, position, normal, force, and speed of the collision are passed to the callback function.

This program will display collision information on screen from the last collision that occurred:

```
#include "engine.h"

char collisioninfo[256];

void _stdcall EntityCollisionCallback( TEntity entity0, TEntity entity1, byte*
position, byte* normal, byte* force, float speed )
{
    TVec3 vecPosition; memcpy(&vecPosition, position, sizeof(TVec3));
    sprintf(collisioninfo, "%d collided with %d at %f,%f,%f", (int)entity0,
(int)entity1, vecPosition.X, vecPosition.Y, vecPosition.Z);
}

int main(int argc, char** argv)
```

```

{
    Initialize();

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

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

    //Create a camera
    TCamera cam=CreateCamera();
    CameraClearColor(cam,Vec4(0,0,1,1));
    MoveEntity(cam,Vec3(0,0,-5));

    //Create a light
    TLight light=CreateDirectionalLight();
    RotateEntity(light,Vec3(65,45,0));

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

    //Create a physics body
    TBody body=CreateBodyBox();
    SetBodyMass(body,1);
    TMesh mesh=CreateCube();
    EntityParent(mesh,body);
    SetEntityKey(body,"name","body1");
    SetEntityCallback(body,(byte*)EntityCollisionCallback,ENTITYCALLBACK_COLLISION);
;

    //Create the ground
    TBody ground=CreateBodyBox(10,0.1,10);
    TMesh groundmesh=CreateCube();
    ScaleEntity(groundmesh,Vec3(10,0.1,10));
    EntityParent(groundmesh,ground);
    PositionEntity(ground,Vec3(0,-1,0));
    SetEntityKey(ground,"name","ground");

    //Enable collision
    Collisions(1,1,1);
    EntityType(body,1);
    EntityType(ground,1);

    int index=0;
    int bodynum=1;
    char temp[100];

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

        index=index+1;
        if (index==30) {
            index=0;
            bodynum+=1;
            body=CopyEntity(body);
            PositionEntity(body,Vec3(0,10,0));
            sprintf(temp,"body%d",bodynum);
            SetEntityKey(body,"name",temp);
        }
    }
}

```

```

        //Update the world
        UpdateWorld();

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

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

        DrawText(0,0,collisioninfo);

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

    exitapp:
    return Terminate();
}

```

### No-response Collision

We can set collision responses so that the collision data is generated, but no physical collision occurs. To do this in the previous program, find this line in the code:

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

Change it to this:

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

Run the program. Collision data will appear on screen, but the bodies will pass right through each other. No-response collision can be used to make an invisible trigger zone the play activates by touching.

### Raycasting

Raycasts are line intersection tests the programmer can perform at any time. Raycasts only work on meshes and terrain. There are two main types of raycasts: visibility tests and picks.

Visibility tests simply return 1 or 0 depending on the result of the test. These are faster because they will abort the raycasting routine as soon as one hit is detected. There are two visibility test functions. The PointVisible function tests visibility between two points in global space:

```
int PointVisible(TVec3 &p1, TVec3 &p2, float radius=0.0, int collisionType=0, Filter filter=0)
```

The EntityVisible function tests visibility between two entity's global positions. Internally, this function just calls PointVisible using the entity positions for the points to test:

```
int EntityVisible(TEntity e1, TEntity e2, float radius=0, int collisionType=0, Filter filter=0)
```

The function declaration might look intimidating because of all the optional parameters, but it is actually quite simple. All pick commands have a few optional parameters:

### Radius

A radius value can be specific to use a thick ray. The default value is 0.0, resulting in a 1-dimensional ray.

### CollisionType

A collision type can be specified, and the ray will only intersect with objects that have a defined collision response with this type. It may be confusing that collision responses can be used with raycasts, because until now we have maintained that collision and raycasting were entirely different techniques.

However, the collision response behavior is very convenient to use, even if it does mix our paradigms up a bit.

### Filter

The filter callback provides one additional method with which raycasts can be controlled. For every object tested, the filter callback will be called. If the filter callback returns 1, the entity will be tested. If the filter callback returns 0 the entity will be skipped. The filter callback must use the following syntax:

```
int Filter(TEntity entity)
```

The pick functions will test all entities until the closest intersecting entity is found. Because picks do not use the "early-out" method that visibility tests use, they can be much slower. The basic pick function is LinePick:

```
int LinePick(TVec3 &p1, TVec3 &p2, float radius=0.0, int collisionType=0, Filter filter=0)
```

The function declaration is identical to the PointVisible function except that a pick object is accepted. A pick object stores all information about the pick result. If the function returns one, the pick values will be filled into the supplied pick object. The structure for the pick class is as follows, and contains the picked entity, surface, position, and normal. Note that the surface field may be Null if the picked entity is not a mesh:

```
struct TPick
{
    TEntity entity;
    TSurface surface;
    float X;
    float Y;
    float Z;
    float NX;
    float NY;
    float NZ;
}
```

All other pick functions internally call the LinePick function. The EntityPick will perform a pick test projecting out in the direction an entity is facing on its z axis:

```
int EntityPick(TPick *pick, TEntity e, float range, float radius=0.0, int collisionType=0, Filter filter=0)
```

The CameraPick function will perform a raycast between a camera position and screen coordinates transformed to global space:

```
int CameraPick(TPick *pick, TEntity camera, TVec3 &p, float radius=0.0, int collisionType=0, Filter filter=0)
```

## Raycasting in Action

The following program will perform a visibility test between two points. The arrow keys can be used to move the occluding cube up and down:

```
#include "engine.h"

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

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

```

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

//Create a camera
TCamera cam=CreateCamera();
CameraClearColor(cam, Vec4(0,0,1,1));
MoveEntity(cam, Vec3(0,0,-5));

//Create a light
TLight light=CreateDirectionalLight();
RotateEntity(light, Vec3(65,45,0));

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

//Create a mesh
TMesh cube=CreateCube();

char temp[100];
int result;
TVec3 point0;
TVec3 point1;

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

    //Move the cube with arrow keys
    if (KeyDown(KEY_UP)) { MoveEntity(cube, Vec3(0,0.01,0)); }
    if (KeyDown(KEY_DOWN)) { MoveEntity(cube, Vec3(0,-0.01,0)); }

    //Update the world
    UpdateWorld();

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

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

    //Test and display visibility between two points
    result=PointVisible(Vec3(4,0,0), Vec3(-4,0,0));
    sprintf(temp, "Visible: %d", result);
    DrawText(0,0,temp);

    //Display the tested line
    point0=CameraUnproject(cam, Vec3(4,0,0));
    point1=CameraUnproject(cam, Vec3(-4,0,0));
    SetColor(Vec4(1,0,0,1));
    DrawLine(point0.X, point0.Y, point1.X-point0.X, point1.Y-point0.Y);
    SetColor(Vec4(1));

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

exitapp:
return Terminate();
}

```

This program will perform a pick test. Unlike the visibility test, the pick function provides us with the closest intersection point:

```
#include "engine.h"

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

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

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

    //Create a camera
    TCamera cam=CreateCamera();
    CameraClearColor(cam,Vec4(0,0,1,1));
    MoveEntity(cam,Vec3(0,0,-5));

    //Create a light
    TLight light=CreateDirectionalLight();
    RotateEntity(light,Vec3(65,45,0));

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

    //Create a mesh
    TMesh cube=CreateCube();

    char temp[100];
    int result;
    TVec3 point0;
    TVec3 point1;
    TPick pick;

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

        //Move the cube with arrow keys
        if (KeyDown(KEY_UP)) { MoveEntity(cube,Vec3(0,0.01,0)); }
        if (KeyDown(KEY_DOWN)) { MoveEntity(cube,Vec3(0,-0.01,0)); }

        //Update the world
        UpdateWorld();

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

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

        //Test and display visibility between two points
        if (LinePick(&pick,Vec3(4,0,0),Vec3(-4,0,0)) {
            //Display the tested line
            point0=CameraUnproject(cam,Vec3(4,0,0));
```

```

        point1=CameraUnproject(cam,Vec3(pick.X,pick.Y,pick.Z));
        SetColor(Vec4(1,0,0,1));
        DrawLine(point0.X,point0.Y,point1.X-point0.X,point1.Y-point0.Y);
        SetColor(Vec4(1));
        sprintf(temp,"Picked position: %f,%f,%f",pick.X,pick.Y,pick.Z);
        DrawText(0,0,temp);
    }
    else {
        point0=CameraUnproject(cam,Vec3(4,0,0));
        point1=CameraUnproject(cam,Vec3(-4,0,0));
        SetColor(Vec4(1,0,0,1));
        DrawLine(point0.X,point0.Y,point1.X-point0.X,point1.Y-point0.Y);
        SetColor(Vec4(1));
    }

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

exitapp:
return Terminate();
}

```

## Smarter Raycasting

Now we are going to learn how to dismiss select entities from raycast tests. The program below displays three meshes and perform a visibility test that intersects all three:

```

#include "engine.h"

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

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

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

    //Create a camera
    TCamera cam=CreateCamera();
    CameraClearColor(cam,Vec4(0,0,1,1));
    MoveEntity(cam,Vec3(0,0,-5));

    //Create a light
    TLight light=CreateDirectionalLight();
    RotateEntity(light,Vec3(65,45,0));

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

    //Create a mesh
    TMesh cube=CreateCube();
}

```

```

//Create 2 spheres
TMesh mesh1=CreateSphere();
PositionEntity(mesh1,Vec3(2,0,0));
TMesh mesh2=CreateSphere();
PositionEntity(mesh2,Vec3(-2,0,0));

char temp[100];
int result;
TVec3 point0;
TVec3 point1;

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

    //Move the cube with arrow keys
    if (KeyDown(KEY_UP)) { MoveEntity(cube,Vec3(0,0.01,0)); }
    if (KeyDown(KEY_DOWN)) { MoveEntity(cube,Vec3(0,-0.01,0)); }

    //Update the world
    UpdateWorld();

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

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

    //Test and display visibility between two points
    result=PointVisible(Vec3(4,0,0),Vec3(-4,0,0));
    sprintf(temp,"Visible: %d",result);
    DrawText(0,0,temp);

    //Display the tested line
    point0=CameraUnproject(cam,Vec3(4,0,0));
    point1=CameraUnproject(cam,Vec3(-4,0,0));
    SetColor(Vec4(1,0,0,1));
    DrawLine(point0.X,point0.Y,point1.X-point0.X,point1.Y-point0.Y);
    SetColor(Vec4(1));

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

exitapp:
return Terminate();
}

```

Let's assume we want to perform the visibility test on only the cube. There are several ways to achieve this. The first is by hiding the unwanted entities when the visibility test is performed. When we move the cube out of the way of the ray, the visibility test will return 1:

```

#include "engine.h"

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

```

```

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

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

//Create a camera
TCamera cam=CreateCamera();
CameraClearColor(cam,Vec4(0,0,1,1));
MoveEntity(cam,Vec3(0,0,-5));

//Create a light
TLight light=CreateDirectionalLight();
RotateEntity(light,Vec3(65,45,0));

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

//Create a mesh
TMesh cube=CreateCube();

//Create 2 spheres
TMesh mesh1=CreateSphere();
PositionEntity(mesh1,Vec3(2,0,0));
TMesh mesh2=CreateSphere();
PositionEntity(mesh2,Vec3(-2,0,0));

char temp[100];
int result;
TVec3 point0;
TVec3 point1;

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

    //Move the cube with arrow keys
    if (KeyDown(KEY_UP)) { MoveEntity(cube,Vec3(0,0.01,0)); }
    if (KeyDown(KEY_DOWN)) { MoveEntity(cube,Vec3(0,-0.01,0)); }

    //Update the world
    UpdateWorld();

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

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

    //Test and display visibility between two points
    HideEntity(mesh1);
    HideEntity(mesh2);
    result=PointVisible(Vec3(4,0,0),Vec3(-4,0,0));
    ShowEntity(mesh1);
    ShowEntity(mesh2);
    sprintf(temp,"Visible: %d",result);
    DrawText(0,0,temp);
}

```

```

        //Display the tested line
        point0=CameraUnproject(cam,Vec3(4,0,0));
        point1=CameraUnproject(cam,Vec3(-4,0,0));
        SetColor(Vec4(1,0,0,1));
        DrawLine(point0.X,point0.Y,point1.X-point0.X,point1.Y-point0.Y);
        SetColor(Vec4(1));

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

    exitapp:
    return Terminate();
}

```

However, it is not convenient to hide and show individual entities each time a visibility test is performed in a complex application. If we assign a collision type to the cube, enable collisions between type one and one, and specify a collision type for the visibility test, the test will only be performed on the cube mesh:

```

#include "engine.h"

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

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

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

    //Create a camera
    TCamera cam=CreateCamera();
    CameraClearColor(cam,Vec4(0,0,1,1));
    MoveEntity(cam,Vec3(0,0,-5));

    //Create a light
    TLight light=CreateDirectionalLight();
    RotateEntity(light,Vec3(65,45,0));

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

    //Create a mesh
    TMesh cube=CreateCube();
    Collisions(1,1,true);
    EntityType(cube,1);

    //Create 2 spheres
    TMesh mesh1=CreateSphere();
    PositionEntity(mesh1,Vec3(2,0,0));
    TMesh mesh2=CreateSphere();
    PositionEntity(mesh2,Vec3(-2,0,0));

    char temp[100];
}

```

```

int result;
TVec3 point0;
TVec3 point1;

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

    //Move the cube with arrow keys
    if (KeyDown(KEY_UP)) { MoveEntity(cube,Vec3(0,0.01,0)); }
    if (KeyDown(KEY_DOWN)) { MoveEntity(cube,Vec3(0,-0.01,0)); }

    //Update the world
    UpdateWorld();

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

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

    //Test and display visibility between two points
    result=PointVisible(Vec3(4,0,0),Vec3(-4,0,0),0.0,1);
    sprintf(temp,"Visible: %d",result);
    DrawText(0,0,temp);

    //Display the tested line
    point0=CameraUnproject(cam,Vec3(4,0,0));
    point1=CameraUnproject(cam,Vec3(-4,0,0));
    SetColor(Vec4(1,0,0,1));
    DrawLine(point0.X,point0.Y,point1.X-point0.X,point1.Y-point0.Y);
    SetColor(Vec4(1));

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

exitapp:
return Terminate();
}

```

Finally, we can dismiss unwanted intersections using the filter callback. We'll add this function before the main function:

```

int PickFilter( TEntity entity ) {
    if (entity==cube) {
        return 1;
    }
    else {
        return 0;
    }
}

```

Then we can specify the PickFilter callback in the call to PointVisible:

```
#include "engine.h"

TMesh cube;

int _stdcall PickFilter( TEntity entity ) {
    if (entity==cube) {
        return 1;
    }
    else {
        return 0;
    }
}

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

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

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

    //Create a camera
    TCamera cam=CreateCamera();
    CameraClearColor(cam,Vec4(0,0,1,1));
    MoveEntity(cam,Vec3(0,0,-5));

    //Create a light
    TLight light=CreateDirectionalLight();
    RotateEntity(light,Vec3(65,45,0));

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

    //Create a mesh
    cube=CreateCube();

    //Create 2 spheres
    TMesh mesh1=CreateSphere();
    PositionEntity(mesh1,Vec3(2,0,0));
    TMesh mesh2=CreateSphere();
    PositionEntity(mesh2,Vec3(-2,0,0));

    char temp[100];
    int result;
    TVec3 point0;
    TVec3 point1;

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

        //Move the cube with arrow keys
        if (KeyDown(KEY_UP)) { MoveEntity(cube,Vec3(0,0.01,0)); }
        if (KeyDown(KEY_DOWN)) { MoveEntity(cube,Vec3(0,-0.01,0)); }
    }
}
```

```

        //Update the world
        UpdateWorld();

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

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

        //Test and display visibility between two points
        result=PointVisible(Vec3(4,0,0),Vec3(-4,0,0),0.0,0,(BP)PickFilter);
        sprintf(temp,"Visible: %d",result);
        DrawText(0,0,temp);

        //Display the tested line
        point0=CameraUnproject(cam,Vec3(4,0,0));
        point1=CameraUnproject(cam,Vec3(-4,0,0));
        SetColor(Vec4(1,0,0,1));
        DrawLine(point0.X,point0.Y,point1.X-point0.X,point1.Y-point0.Y);
        SetColor(Vec4(1));

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

    exitapp:
    return Terminate();
}

```

## Camera Picking

We can pick objects using the screen coordinates. We'll create a red material to signify which object is selected:

```

#include "engine.h"

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

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

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

    //Create a camera
    TCamera cam=CreateCamera();
    CameraClearColor(cam,Vec4(0,0,1,1));
    MoveEntity(cam,Vec3(0,0,-5));

    //Create a light
    TLight light=CreateDirectionalLight();
}

```

```

RotateEntity(light,Vec3(65,45,0));

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

//Create a mesh
TMesh cube=CreateCube();

//Create 2 spheres
TMesh mesh1=CreateSphere();
PositionEntity(mesh1,Vec3(2,0,0));
TMesh mesh2=CreateSphere();
PositionEntity(mesh2,Vec3(-2,0,0));

char temp[100];
int result;
TVec3 point0;
TVec3 point1;
TPick pick;

//Create materials
TMaterial selectionmaterial=CreateMaterial();
SetMaterialColor(selectionmaterial,Vec4(1,0,0,1));

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

    //Pick objects with mouse
    if (MouseHit(1)) {
        PaintEntity(cube,0);
        PaintEntity(mesh1,0);
        PaintEntity(mesh2,0);
        if (CameraPick(&pick,cam,Vec3(MouseX(),MouseY(),1000))) {
            PaintEntity(pick.entity,selectionmaterial);
        }
    }

    //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();
}

```

Copyright Leadwerks Software 2008-2009

All Rights Reserved.