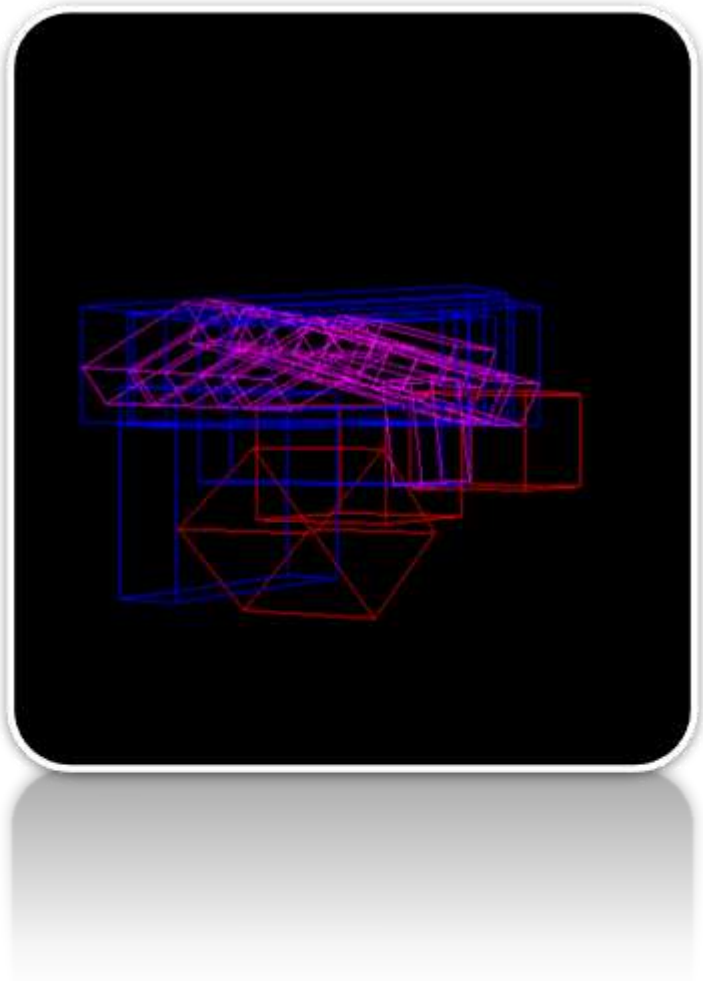
Grab Library

User Manual



Introduction

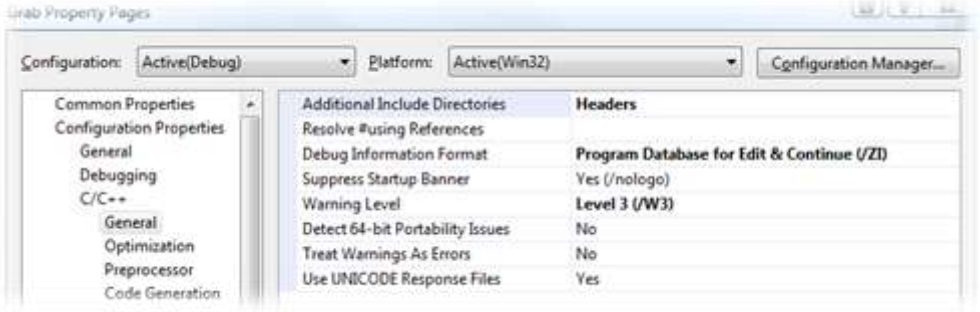
The aim of this manual is to instruct a developer on how to use the Grab library and what they can do with it. It will inform you on how to install and use the library during development, give details on its functionality and provide examples of its uses.

Installation

VC 2005/2008

To use the library in a Visual C++ 2005 or 2008 environment, you can either use the .lib files provided for each or the original source code if you wish to modify or compile it yourself.

To use the .lib file, it is recommended that you create a folder 'Headers' in your project solution with which you wish to use this library and copy them to there. You will next go to the properties of the project you are working and add the Headers directory to the field under 'Configuration Properties > C/C++ > General entitled 'Additional Include Directories'. To ensure you can user this on other systems, it is recommended that you make paths such as these relative. One way of doing this is to put the folder you wish to link to in the main solution directory and then simply put the name of the folder in the required field.



Using the library

Constructing a Grab

The grab consists of a limb which performs the grabbing and an object being grabbed stored in a single class which overlooks the construction, maintenance, testing and animation of both.

The following section will describe the process of creating a Grab situation with the library, as made in the 'OpenGLTest Functional' example application and solution project. The solution can be found in the 'Solutions\Visual Studio 2008 Solutions\Testing' in which it is entitled 'OpenGLTest Functional'. If you wish to view it outside of visual studio, you can just use the source code OpenGLTest_Functional.cpp with the Grab library.

First, we create our Grab class.

```
Grab *GRAB;
```

Then we create some OBBs (Object Bounding Boxes) to describe the geometry of the object being grabbed. You can add multiple object geometries to the grab class to describe a single object, giving an individual weight and dimension to each.

```
OBB *ObjectGeometry_OBB1 = new OBB;
ObjectGeometry_OBB1->SetMinMax(VectorF(0.0f,0.0f,0.0f),
VectorF(1.0f,1.0f,1.0f));
```

Then we make some geometric objects to assign these OBBs to and add to the Grab class.

```
GObject *GrabObject1 = new GObject;
GrabObject1->SetOBB(ObjectGeometry_OBB1);
GrabObject1->SetWeight(1.0f);
GRAB->AddObjectGeometry(GrabObject1);
```

Next, we can make the Hand so as to describe the Limb grabbing the object. Here, we have also set its strength. This is the base strength of the hand irrespective of the other segments to be added later.

```
Hand *AHand = new Hand;
AHand->SetStrength(5.0f);
```

We can not begin describing the geometry of the Hand. First, with the palm, onto which the fingers are later positioned and at which the object by default will move. The palm's OBB is set here via it's minimum and maximum boundings.

```
OBB *PalmOBB = new OBB;
PalmOBB->SetMinMax(VectorF(0.0f,0.0f,0.0f), VectorF(2.0f,2.0f,0.5f));
AHand->SetPalmOBB(PalmOBB);
```

Now that we have a palm we can start creating finger segments from which to construct fingers. Here, as with all entities, we first create its bounding volume in the form of an OBB.

```
OBB  GenericFingerSegmentOBB = new OBB;
GenericFingerSegmentOBB->SetMinMax(VectorF(0.0f,0.0f,0.0f),VectorF(0.4f,1.0f,0.4f));
```

The generic finger segment can now be used to construct other finger segments to be used as bases on the fingers. First, the finger segment required is created. Here, we see the base segment of the 'little finger' created. Fingers can be of any configuration and name. The construction of a hand based on that of a human has merely been undertaken to be relatable.

```
OBB *LittleFinger_OBB1 = new OBB(GenericFingerSegmentOBB);
FingerSegment *LittleFinger_Segment1 = new FingerSegment;
LittleFinger_Segment1->SetOBB(LittleFinger_OBB1);
```

The segment can then be rotated or translated to better resemble the intended shape as well as having its strength set.

```
LittleFinger_Segment1->Rotate(VectorF(0.0f, 0.0f, 0.125f));
LittleFinger_Segment1->SetStrength(0.2f);
```

The finger is now built. It can be constructed of any number of segments but they will all be translated so that the centre of their base coordinates is at the centre of the tip of the previously added segment. As can be seen in the following example, even the same segment can be added. This is because the finger will position a new segment of its parameters in the correct next position. A

strength value is also set. It is not necessary to set a strength value but it will otherwise add nothing to the total strength of the limb during a grab.

```
Finger *LittleFinger = new Finger;
LittleFinger->AddFingerSegment(LittleFinger_Segment1);
LittleFinger->AddFingerSegment(LittleFinger_Segment1);
LittleFinger->AddFingerSegment(LittleFinger_Segment1);
LittleFinger->SetStrength(0.05f);
```

The finger is finally added to the hand, at a provided position. Here we can also see an ID being acquired from the addition. This can be retrieved from any of the Add functions, meaning you can access a specific segment's data at any point from outside of the Grab class through the Get functions that take an ID.

The limb is also created and the hand added to it.

```
LittleFingerID = AHand->AddFinger(LittleFinger, VectorF(0.0f,2.0f,0.0f));
Limb *ALimb = new Limb;
ALimb->AddHand(AHand);
```

There is only one limb to be involved in each grab event and as such the function is only a SetLimb function:

```
GRAB->SetLimb(ALimb);
```

Using a Grab

It is now assumed you have successfully created an instance of a Grab object and have all the relevant data in place to describe your limb and object. Drawing again from the OpenGL demo source code, we will now go through the process of undertaking a grab event and retrieving information from it.

Moving Objects

The first stage to be undergone in the demo is to bring the object towards the hand until it collides. This is handled in the library by the function *IterateMoveObjects*(). This function moves all of the object geometry in a manner defined either by the user or by its default means. The default action, as employed in the demo, is to move all of the objects towards the centre of the palm, as is most likely to be the intention of a character.

Other options available are to set a destination with *SetObjectDestination(VectorF *destination)* or set a direction for the objects to move with *SetObjectDirection(VectorF *direction).*

As employed in the demo, the *HandCollided()* function can be used to detect if any segment of the hand has collided with any geometric component of the object so as to stop moving as soon as they touch.

Closing Hand

Once the objects are at a satisfactory position the hand can then be closed around the object until one of two events have occurred:

1. All bendable segments of the limb have reached their limit
2. All segments, or at least their end segments, have collided with some geometry of the object.

Retrieving Information

In the demo, the information retrieved is as follows:

- The position, orientation and bounding volumes of the segments and geometry is used to render the visual output in methods such as the following in which the OBB of the little finger is acquired:

```
VIZ->DrawCube(GRAB->GetFinger(LittleFingerID)->GetOBB()->Vertices(), 0.0f, 0.0f, 1.0f);
```

Or the individual base segment acquired via the following:

```
VIZ->DrawCube(GRAB->GetFinger(LittleFingerID)->GetFingerSegment(0)->GetOBB()->Vertices(), 1.0f, 0.0f, 1.0f);
```

GetAABB() and GetSphere functions are also available for all items in grab that are collideable.

- At the end of the second stage of the grab, when the hand encloses the object, the following code produces some information to the console window:

```
cout << "\nNumber of Collided Finger Segments: " << GRAB->NumCollidedFingerSegments() << "\nTotal Strength of Collided Limb: " << GRAB->TotalStrengthCollided() <<"\nTotal weight of object: " << GRAB->TotalWeight();
cout << "\nMove Iterations: " << GRAB->MoveIterations();
cout << "\nRotation Iterations: " << GRAB->RotationIterations();
```

Here, we see the functions available to acquire:

1. The number of collided finger segments (number of hands and fingers is also possible)
2. The total strength of all successfully collided segments of the limb
3. The total weight of all object geometry
4. The number of iterations that occurred in moving the objects
5. The number of iterations that occurred in bending the finger segments