

ODE for newbies

(especially those from a Tokamak background!)

Written by David Coombes

Introduction

I was writing a software application using the free Tokamak physics library. It was very easy to get up and running and worked very well. Unfortunately when taken to a much slower PC it could only handle a few objects on screen. It was time for a change of library.

After a brief look at the JV-ODE demos, and seeing how much quicker it was generally, even without it's useful speed optimisations like auto-deactivation, it seemed an essential acquisition.

Of course, it works differently to Tokamak, and offered a few conceptual stumbling blocks until I could get it to work. Now that I have, I'll share my insight so hopefully you'll find it easier to work with. I'm not going into great detail on the functions, but providing a quick-start guide on the basics. These same steps/principles will apply to probably every ODE program you ever write.

The Nature of an ODE object

ODE describes objects inside it's engine using 2 'classes', geometry, and body.

Geometry defines the collision surface of an object.

Body defines the physical body that is subject to forces, with a mass and centre of gravity.

A geometry is bound to a body, so when forces are applied to the body, the geometry moves, and when the geometry collides, the body reacts.

With this little scrap of info we can create our first ODE program.

The Nature of ODE collisions

There's quite a conceptual difference between Tokamak and ODE when it comes to collisions I think, and it's certainly worth explaining up front. ODE is based on the principle of **joints**. Joints define a positional relationship between objects. You can create hinge joints and ball-and-socket joints to limit the motion of one object (ODE body) relative to another. ODE's collisions also use joints. Each collision that occurs creates a joint of type contact between the colliding surfaces. This is done by the function

```
dSpaceCollide(space,world,contactGroup)
```

These creates a list of all the joints of type contact – a list of all the collisions. Each contact joint can have it's own properties to determine the resolved forces. You can set friction and restitution (bounce) for a contact.

This is a very versatile system, giving excellent control over all sorts of effects. But for the most part simple physics with defined materials can use some generic code which I'll include in this tutorial.

First code

Starting with the basics of a Blitz3D program.

```
Include "ode.bb"  
Graphics3D 800,600,0,2
```

“ode.bb” is the file of that name that came with the wrapper. The first bit of ODE code to add creates a working space for ODE.

```
Global ODE_world=dWorldCreate()  
Global ODE_space=dHashSpaceCreate(0)  
Global ODE_contactGroup=dJointGroupCreate(0)
```

This basically reserves space in RAM for storing essential information for the physics engine. The next step is to arrange all our object information into a convenient type.

```
Type ODE_object  
; An ODEGeom object contains a body, collision geom, and pointer to Blitz mesh  
Field body  
Field geom  
Field mesh  
End Type
```

This definition has **body** and **geom** fields as handles for the ODE objects, and also a **mesh** field as a handle for a Blitz mesh. With definitions out the way, we initialise the ODE engine

```
dWorldSetAutoDisableFlag(ODE_world,1) ; Turn on auto disable  
dWorldSetGravity(ODE_world,0,-1,0) ; Gravity in m/s/s  
dWorldSetERP(ODE_world,0.2) ; Global 'error correction'
```

These lines enable auto-disabling, set the gravity, and global error correction. In order to see our objects we'll need a camera and light, so let's get that out the way...

```
; Create light  
light1=CreateLight()  
RotateEntity light1,45,-90,0  
LightColor light1,255,255,255  
AmbientLight 130,130,130  
  
; Create camera  
cam=CreateCamera()  
PositionEntity cam,0,60,-170  
TurnEntity cam,0,15,0
```

Let's now create some ODE objects.

```
obj.ODE_object=New ODE_object  
  
; Create a physics body and specify which world it occupies  
obj\body=dBodyCreate(ODE_world)  
  
; Position and orientation of physics body  
dBodySetRotation(obj\body,0,0,0)  
dBodySetPosition(obj\body,0,100,0)  
  
; Enable auto-disabling  
dBodySetAutoDisableFlag(obj\body,1)
```

```

; Create collision geometry for the created physics body. This takes the position+orientation of the
body by default
obj\geom=dCreateBox(ODE_space,10,10,10)

; Bind geom to body
dGeomSetBody(obj\geom,obj\body)

; create a Blitz mesh so we can see the box
obj\mesh=CreateCube()

ScaleMesh obj\mesh,5,5,5

```

The above works in three steps. Step 1 creates a body, which we give no rotation and position 100 units up in the air. Step 2 creates collision geometry that defines the physical surfaces that will bash into things. Step 3 creates a Blitz box that we will place to fit the ODE body. Note that the dimensions of the mesh are half those of the ODE geom, as Blitz and ODE scale boxes differently. Note also the use of our previously defined **ODE_World** and **ODE_Space** variables as to where we create these items. ODE allows more than one world and space, but generally everything you want will be happening in the same space.

That's created a box. What we need now is a floor, or gravity will cause our box to fall forever.

```

; Create a floor
ground.ODE_object=New ODE_Object
ground\geom=dCreatePlane(ODE_space,0,1,0,0)
ground\mesh=CreatePlane() ; Blitz mesh
EntityColor ground\mesh,120,120,120

```

Very similar to creating a box, we create a new ODE_object (custom type) and add geometry and a mesh. Different to the box, we don't create a body. A body is used for controlling movement in response to forces, and as a floor doesn't move it needs no body.

So, our ODE objects are created and it's time to see them in action. Here comes the main loop.

```

While Not KeyHit(1)

; Perform collisions, supplying the world, space, and collision data
dSpaceCollide(ODE_space,ODE_world,ODE_contactGroup)

; Advance the simulation
dWorldQuickStep(ODE_world,0.1)

; Clear collision data
dJointGroupEmpty(ODE_contactGroup)

UpdateWorld
RenderWorld

Flip

Wend

```

You'll know **UpdateWorld**, **RenderWorld** and **Flip** from Blitz. The new components in the above are

```
dSpaceCollide(ODE_space,ODE_world,ODE_contactGroup)
```

This finds all the collisions in the ODE space, in the ODE world, and records it in the ODE_contact group so we can look them up later if we want.

```
dWorldQuickStep(ODE_world,0.1)
```

This advances the simulation of the ODE world by 0.1 steps. Decreasing this value causes smaller time steps between frames, slowing down motion with improved accuracy. Advancing the simulation uses the information in **ODE_contactGroup** to control collisions and responses.

```
dJointGroupEmpty(ODE_contactGroup)
```

This function clears the collision information, performed after the simulation has been updated.

After the loop, we need to clear away all the ODE information

```
; Destory all the ODE stuff
dJointGroupDestroy(ODE_contactGroup)
dSpaceDestroy(ODE_space)
dWorldDestroy(ODE_world)
dCloseODE()

End
```

Now if you run this program, you will see a box sat on the floor. What we haven't done yet is match the visible Blitz mesh to the position and orientation of the box body in the ODE simulation. We need to add this code into our main loop.

```
While Not KeyHit(1)

  For ode.ODE_object=Each ODE_object
    If ode\body
      If dBodiesEnabled(ode\body)
        RotateEntity
          ode\mesh,dGeomGetPitch#(ode\geom),dGeomGetYaw#(ode\geom),dGeomGetRoll#(ode\geom)
        PositionEntity
          ode\mesh,dGeomGetPositionX#(ode\geom),dGeomGetPositionY#(ode\geom),dGeomGetPositionZ#(ode\geom)
      End If
    End If
  Next
```

It's just the same as Tokamak. For every ODE_object we've created, position and orientate it's mesh to the ODE body. We do a couple of checks. First to see if the object has an ODE body, because if it doesn't we can't match the mesh to the body! And then a check to see if the object is enabled (taking part in the simulation). Disabling of bodies is a great optimisation that Tokamak was lacking, as if an object is stationary there's no need to calculate collision and such for it.

More boxes please!

Of course there's little point in having a complex physics engine if you're just going to have one box on a floor! Lets create multiple boxes by writing an ODE_Box function. Using the code from the box creation we can write...

```
Function create_Box.ODE_object(x_pos, y_pos, z_pos, x_siz, y_siz, z_siz, x_rot, y_rot, z_rot)
; Create ODE object to define object's properties
obj.ODE_object=New ODE_object

; Create a physics body and specify which world it occupies
obj\body=dBodyCreate(ODE_world)
```

```

; Positon and orientation of physics body
dBodySetRotation(obj\body, x_rot, y_rot, z_rot)
dBodySetPosition(obj\body, x_pos, y_pos, z_pos)

; Enable auto-disabling
dBodySetAutoDisableFlag(obj\body,1)

; Create collision geometry for the created physics body. This takes the position+orientation of the
body
obj\geom=dCreateBox(ODE_space,x_siz, y_siz, z_siz)
dGeomSetBody(obj\geom,obj\body)

obj\mesh=CreateCube()
ScaleMesh obj\mesh,x_siz/2,y_siz/2,z_siz/2

Return obj
End Function

```

This returns an ODE_Object type. We can then replace our box creation code with...

```

obj.ODE_object=New ODE_object
obj = create_Box(0,100,0,10,10,10,0,0,0)

```

With this function we can easily populate our scene with loads of boxes. Let's create an array of ODE boxes. After the initialisation of ODE we add...

```

Dim boxes.ODE_object(200)
For n=0 To 200
  boxes(n) = New ODE_object
Next

```

Room for 200 boxes. Then we replace our box creation with...

```

For n=0 To 50
; randomise sizes
x#=1+Rnd(0,10)
y#=1+Rnd(0,10)
z#=1+Rnd(0,10)

boxes(n)=create_Box(Rnd(-40,40),100+Rnd(0,600),Rnd(-40,40), x,y,z, Rnd(-30,30),Rnd(-
30,30),Rnd(-30,30))
Next; boxes

```

This creates 51 boxes with random sizes, positions and rotations. The whole program looks like this...

```

Include "ode.bb"
Graphics3D 800,600,0,2

Global ODE_world=dWorldCreate()
Global ODE_space=dHashSpaceCreate(0)
Global ODE_contactGroup=dJointGroupCreate(0)

Type ODE_object
; An ODEGeom object contains a body, collision geom, and pointer to Blitz mesh
Field body
Field geom
Field mesh
End Type

dWorldSetAutoDisableFlag(ODE_world,1) ; Turn on auto disable
dWorldSetGravity(ODE_world,0,-1,0) ; Gravity in m/s/s

```

```

dWorldSetERP(ODE world.0.2) : Global 'error correction'

Dim boxes.ODE_object(200)
For n=0 To 200
    boxes(n) = New ODE_object
Next

; Create light
light1=CreateLight()
RotateEntity light1,45,-90,0
LightColor light1,255,255,255
AmbientLight 130,130,130

; Create camera
cam=CreateCamera()
PositionEntity cam,0,60,-170
TurnEntity cam,0,15,0

For n=0 To 50
    ; randomise sizes
    x#=1+Rnd(0,10)
    y#=1+Rnd(0,10)
    z#=1+Rnd(0,10)

    boxes(n)=create_Box(Rnd(-40,40),100+Rnd(0,600),Rnd(-40,40), x,y,z, Rnd(-30,30),Rnd(-30,30),Rnd(-30,30))
Next; boxes

; Create a floor
ground.ODE_object=New ODE_Object
ground\geom=dCreatePlane(ODE_space,0,1,0,0)
ground\mesh=CreatePlane() ; Blitz mesh
EntityColor ground\mesh,120,120,120

While Not KeyHit(1)

    ; UpdateGeoms() - convenient to put into a function but shown here in the main loop
    For ode.ODE_object=Each ODE_object
        If ode\body
            If dBodyIsEnabled(ode\body)
                RotateEntity
                ode\mesh,dGeomGetPitch#(ode\geom),dGeomGetYaw#(ode\geom),dGeomGetRoll#(ode\geom)
                PositionEntity
                ode\mesh,dGeomGetPositionX#(ode\geom),dGeomGetPositionY#(ode\geom),dGeomGetPositionZ#(ode\geom)
            End If
        End If
    Next

    ; Perform collisions, supplying the world, space, and collision data
    dSpaceCollide(ODE_space,ODE_world,ODE_contactGroup)

    ; Advance the simulation
    dWorldQuickStep(ODE_world,0.1)

    ; Clear collision data
    dJointGroupEmpty(ODE_contactGroup)

    UpdateWorld
    RenderWorld

    Flip

Wend

```

```

: Destory all the ODE stuff
dJointGroupDestroy(ODE_contactGroup)
dSpaceDestroy(ODE_space)
dWorldDestroy(ODE_world)
dCloseODE()

End

Function create_Box.ODE_object(x_pos, y_pos, z_pos, x_siz, y_siz, z_siz, x_rot, y_rot, z_rot)
; Create ODE object to define object's properties
obj.ODE_object=New ODE_object

; Create a physics body and specify which world it occupies
obj\body=dBodyCreate(ODE_world)

; Positon and orientation of physics body
dBodySetRotation(obj\body, x_rot, y_rot, z_rot)
dBodySetPosition(obj\body, x_pos, y_pos, z_pos)

; Enable auto-disabling
dBodySetAutoDisableFlag(obj\body,1)

; Create collision geometry for the created physics body. This takes the position+orientation of the
body
obj\geom=dCreateBox(ODE_space,x_siz, y_siz, z_siz)
dGeomSetBody(obj\geom,obj\body)

obj\mesh=CreateCube()
ScaleMesh obj\mesh,x_siz/2,y_siz/2,z_siz/2

Return obj
End Function

```

If you run it, you'll get 51 assorted boxes falling from the sky, hitting the floor, and bashing into each other. You'll maybe notice a lot of slipping sliding about. We can change the degree of global friction by adding this command when we set up ODE...

```
dContactSetMu(10.0)
```

This sets the coefficient of friction. A value of 0.0 means no friction – everything slides forever. Bigger numbers mean more friction.

If you're coming from Tokamak, you'll think of physics in terms of Rigid Bodies that are moved and collide within the engine, and Animated Bodies that collide but don't move with the simulation. An Animated Body would be used for walls that don't get knocked about, with Rigid Bodies being the objects moving around in the room. ODE doesn't have these distinctions, but you can create non-moving collision objects as we did with the floor. Let's add a box function that creates a collidable box which doesn't move...

```

Function create_Rigid_Box.ODE_object(x_pos, y_pos, z_pos, x_siz, y_siz, z_siz, x_rot, y_rot, z_rot)
; Create ODE object to define object's properties
obj.ODE_object=New ODE_object

; Create collision geometry for the created physics body. This takes the position+orientation of the
body
obj\geom=dCreateBox(ODE_space,x_siz, y_siz, z_siz)

; Positon and orientation of physics body
dGeomSetRotation(obj\geom, x_rot, y_rot, z_rot)
dGeomSetPosition(obj\geom, x_pos, y_pos, z_pos)

obj\mesh=CreateCube()

```

```

ScaleMesh obj\mesh.x siz/2.v siz/2.z siz/2

RotateEntity
obj\mesh,dGeomGetPitch#(obj\geom),dGeomGetYaw#(obj\geom),dGeomGetRoll#(obj\geom)
PositionEntity
obj\mesh,dGeomGetPositionX#(obj\geom),dGeomGetPositionY#(obj\geom),dGeomGetPositionZ#(obj\geom)

Return obj
End Function

```

This is very similar to the create_Box function except that we don't waste time creating a body. We create ODE geometry, position and orientate it, create a mesh, and position and orientate that to match the ODE geometry.

A call to this function creates a shelf...

```

; Rigid shelf
shelf.ODE_object=New ODE_object
shelf=create_Rigid_Box(0,60,0, 80,2,80, 0,0,0)
EntityColor shelf\mesh,255,255,128

```

Run the whole thing and witness boxes landing on shelf and floor alike!

What's distinctly missing though is different materials. We need bouncy rubber materials and slippery ice materials. You can set global defaults for these but more usefully, can set it per object...

Adding different materials

A few simple commands, is all that's needed to make slippery or sticky materials, bounce or not materials. First at ODE initialisation you need to specify the type of contact created when objects collide. Though there's lots to choose from, specifying a bounce response seems to be all that's needed in most cases.

```
dContactSetMode(dContactBounce)
```

Then you can set the properties of you objects using

```

dGeomContactSetBounce(geom,restitution)
dGeomContactSetMu(geom,friction)

```

However, these don't work right unless you create a mass for you objects, which needs a little jiggery pokery (but only a little)

Adding mass

Mass isn't just a value for how heavy an ODE object is. Mass has a centre of gravity too. To add mass to an object ODE has a custom data-type, **Mass**.

This code added to the **create_Box()** and **create_Rigid_Box()** functions will add a suitable mass.

```

v=dMassCreate()
dMassSetBox(v,0.01, x_siz, y_siz, z_siz)
dBodySetMass(obj\body,v)

```

v is created as a new mass object. It is given a mass with the command

```
dMassSetBox(v,0.01, x_siz, y_siz, z_siz)
```

The first parameter is the mass object we are changing the properties of. The second is the density. The last three are the sizes of the mass box. By default the centre of gravity is in the centre of this box. You can move the COG but for most objects that's not necessary.

You will see that the mass is assigned to the ODE body of the ODE object we've created. This mass creation is placed after creating the body and geometry.

In our program you can make these modifications...

```
Function create_Box.ODE_object(x_pos, y_pos, z_pos, x_siz, y_siz, z_siz, x_rot, y_rot, z_rot)
; Create ODE object to define object's properties
obj.ODE_object=New ODE_object

; Create a physics body and specify which world it occupies
obj\body=dBodyCreate(ODE_world)

; Position and orientation of physics body
dBodySetRotation(obj\body, x_rot, y_rot, z_rot)
dBodySetPosition(obj\body, x_pos, y_pos, z_pos)

; Enable auto-disabling
dBodySetAutoDisableFlag(obj\body,1)

; Create collision geometry for the created physics body. This takes the position+orientation of the
body
obj\geom=dCreateBox(ODE_space,x_siz, y_siz, z_siz)
dGeomSetBody(obj\geom,obj\body)

v=dMassCreate()
dMassSetBox(v,0.01, x_siz, y_siz, z_siz)
dBodySetMass(obj\body,v)

obj\mesh=CreateCube()
ScaleMesh obj\mesh,x_siz/2,y_siz/2,z_siz/2

Return obj
End Function
```

This adds mass to the objects when created, and

```
; Rigid shelf
shelf.ODE_object=New ODE_object
shelf=create_Rigid_Box(0,60,0, 80,2,80, -20,0,0)
EntityColor shelf\mesh,255,255,128
dGeomContactSetMu(shelf\geom,0.0)
```

...this makes the shelf perfectly slippery and angled. Run the program and objects simply slide off the shelf.

Make this change...

```
; Rigid shelf
shelf.ODE_object=New ODE_object
```

```
shelf=create Rigid Box(0.60,0. 80,2.80. -20,0,0)
EntityColor shelf\mesh,255,255,128
dGeomContactSetMu(shelf\geom,0.0)
dGeomContactSetBounce(shelf\geom,0.9)
```

...and the boxes bounce off. **dGeomContactSetMu()** sets the friction (0 to infinity) and **dGeomContactSetBounce()** sets the bounciness in the range 0.0 to 1.0, representing how much energy is lost between bounces. A value of 1.0 means bouncing forever.

When determining the reaction of two bodies in collision, the values for both surfaces friction and restitution are considered. It is the average of the two surfaces used. So a super bouncy surface and a no bounce surface will result in a moderately bouncy collision.

Rounding off

That's the basics covered. For adding different shaped objects you can reference the ODE docs which cover the available geometry types. They all work the same way though, only with Cylinder or Sphere in the function name (geometry creation, body creation, mass creation) instead of Cube.

I'll finish off with my own functions for creating simple scenes. I'll create a material property system, enable different objects to be added, and assign them their different properties based on their material.

Library file...

Save this as "ODE_Object_Library.bb"

```
Const RESOLUTION_SPHERE = 12
Const RESOLUTION_CYLINDER = 12

; Enumerated types for materials
Const CON_WOOD=1,CON_GOLD=2,CON_RUBBER=3

; Enumerated Types for geometries
Const GEOM_CUBE=1,GEOM_SPHERE=2,GEOM_CYLINDER=3

; Constants for pointer and forces
Global NUM_OBJECTS
Global mat_wood.material, mat_gold.material, mat_rubber.material

Global ODE_world, ODE_space, ODE_contactGroup

Type vector
  Field x#
  Field y#
  Field z#
End Type
Type material
  Field index ; Index for determining type of material
  Field density#
  Field friction#
  Field restitution#
End Type
Type ODE_object
  Field body
  Field geom
  Field mesh
  Field mass
  Field mat.material
End Type
Type actor
```

```

Field aeom          : index of aeometry type
Field pos.vector
Field rot.vector
Field siz.vector
Field mat           ; matieral index
End Type

; Define Material Properties
mat_wood.material=New material
mat_wood\index = CON_WOOD
mat_wood\friction = 10.0
mat_wood\restitution = 0.4
mat_wood\density = 0.9

mat_gold.material=New material
mat_gold\index = CON_GOLD
mat_gold\friction = 3.0
mat_gold\restitution = 0.2
mat_gold\density = 10.0

mat_rubber.material=New material
mat_rubber\index = CON_RUBBER
mat_rubber\friction = 500.0
mat_rubber\restitution = 0.9
mat_rubber\density = 1.5

Function create_Box_Mesh(obj.ODE_object)
  obj\mesh=CreateCube()
  Select obj\mat\index
    Case CON_WOOD
      EntityColor obj\mesh,120,80,40
      EntityShininess obj\mesh,0.2
    Case CON_GOLD
      EntityColor obj\mesh,200,120,40
      EntityShininess obj\mesh,0.95
    Case CON_RUBBER
      EntityColor obj\mesh,70,70,80
      EntityShininess obj\mesh,0.2
  End Select
End Function

Function create_Sphere_Mesh(obj.ODE_object)
  obj\mesh=CreateSphere(RESOLUTION_SPHERE)
  Select obj\mat\index
    Case CON_WOOD
      EntityColor obj\mesh,120,80,40
      EntityShininess obj\mesh,0.2
    Case CON_GOLD
      EntityColor obj\mesh,200,120,40
      EntityShininess obj\mesh,0.95
    Case CON_RUBBER
      EntityColor obj\mesh,70,70,80
      EntityShininess obj\mesh,0.2
  End Select
End Function

Function create_Cylinder_Mesh(obj.ODE_object)
  obj\mesh=CreateCylinder(RESOLUTION_CYLINDER)
  Select obj\mat\index
    Case CON_WOOD
      EntityColor obj\mesh,120,80,40
      EntityShininess obj\mesh,0.2
    Case CON_GOLD
      EntityColor obj\mesh,200,120,40
      EntityShininess obj\mesh,0.95
    Case CON_RUBBER
      EntityColor obj\mesh,70,70,80

```

```

EntityShininess obj\mesh.0.2
End Select
End Function

Function create_Box(obj.ODE_object, x_pos#, y_pos#, z_pos#, width#, length#, depth#, x_rot#,
y_rot#, z_rot#)
obj\body=dBodyCreate(ODE_world)

dMassSetBox(obj\mass,obj\mat\density, width, length, depth)
dBodySetMass(obj\body,obj\mass)

dBodySetRotation(obj\body, x_rot, y_rot, z_rot)
dBodySetPosition(obj\body, x_pos, y_pos, z_pos)
dBodySetAutoDisableFlag(obj\body,1)

obj\geom=dCreateBox(ODE_space,width, length, depth)
dGeomSetBody(obj\geom,obj\body)
dGeomContactSetBounce(obj\geom,obj\mat\restitution)
dGeomContactSetMu(obj\geom,obj\mat\friction)

create_Box_Mesh(obj)
; Blitz box lengths 2x longer than ODE box length
ScaleMesh obj\mesh,width/2,length/2,depth/2
End Function
Function create_Rigid_Box(obj.ODE_object, x_pos#, y_pos#, z_pos#, width#, length#, depth#, x_rot#,
y_rot#, z_rot#)
obj\geom=dCreateBox(ODE_space,width, length, depth)
dGeomSetRotation(obj\geom, x_rot, y_rot, z_rot)
dGeomSetPosition(obj\geom, x_pos, y_pos, z_pos)

; Blitz box lengths 2x longer than ODE box length
create_Box_Mesh(obj)
ScaleMesh obj\mesh,width/2,length/2,depth/2

RotateEntity
obj\mesh,dGeomGetPitch#(obj\geom),dGeomGetYaw#(obj\geom),dGeomGetRoll#(obj\geom)
PositionEntity
obj\mesh,dGeomGetPositionX#(obj\geom),dGeomGetPositionY#(obj\geom),dGeomGetPositionZ#(obj
\geom)

End Function

Function create_Sphere(obj.ODE_object, x_pos#, y_pos#, z_pos#, radius#, x_rot#, y_rot#, z_rot#)
obj\body=dBodyCreate(ODE_world)

dMassSetSphere(obj\mass,obj\mat\density, radius)
dBodySetMass(obj\body,obj\mass)

dBodySetRotation(obj\body, x_rot, y_rot, z_rot)
dBodySetPosition(obj\body, x_pos, y_pos, z_pos)
dBodySetAutoDisableFlag(obj\body,1)

obj\geom=dCreateSphere(ODE_space,radius)
dGeomSetBody(obj\geom,obj\body)
dGeomContactSetBounce(obj\geom,obj\mat\restitution)
dGeomContactSetMu(obj\geom,obj\mat\friction)

create_Sphere_Mesh(obj)
ScaleMesh obj\mesh,radius,radius,radius

End Function
Function create_Rigid_Sphere(obj.ODE_object, x_pos#, y_pos#, z_pos#, radius#, x_rot#, y_rot#,
z_rot#)
obj\geom=dCreateSphere(ODE_space,radius)
dGeomSetRotation(obj\geom, x_rot, y_rot, z_rot)

```

```

dGeomSetPosition(obj\geom, x_pos, y_pos, z_pos)

create_Sphere_Mesh(obj)
ScaleMesh obj\mesh,radius,radius,radius

RotateEntity
obj\mesh,dGeomGetPitch#(obj\geom),dGeomGetYaw#(obj\geom),dGeomGetRoll#(obj\geom)
PositionEntity
obj\mesh,dGeomGetPositionX#(obj\geom),dGeomGetPositionY#(obj\geom),dGeomGetPositionZ#(obj\geom)

End Function

Function create_Cylinder(obj.ODE_object, x_pos#, y_pos#, z_pos#, radius#, length#, x_rot#, y_rot#, z_rot#)
obj\body=dBodyCreate(ODE_world)

; dMassSetSphere(v,obj\mat\density, radius)
dMassSetCylinder(obj\mass,obj\mat\density,1, radius,length)
dBodySetMass(obj\body,obj\mass)

dBodySetRotation(obj\body, x_rot, y_rot, z_rot)
dBodySetPosition(obj\body, x_pos, y_pos, z_pos)
dBodySetAutoDisableFlag(obj\body,1)

obj\geom=dCreateCylinder(ODE_space,radius,length)
dGeomSetBody(obj\geom,obj\body)
dGeomContactSetBounce(obj\geom,obj\mat\restitution)
dGeomContactSetMu(obj\geom,obj\mat\friction)

; Blitz cylinder mesh is 2x longer than ODE cylinder length
create_Cylinder_Mesh(obj)
ScaleMesh obj\mesh,radius,length/2,radius

End Function

Function create_Rigid_Cylinder(obj.ODE_object, x_pos#, y_pos#, z_pos#, radius#,length#, x_rot#, y_rot#, z_rot#)
obj\geom=dCreateCylinder(ODE_space,radius,length)
dGeomSetRotation(obj\geom, x_rot, y_rot, z_rot)
dGeomSetPosition(obj\geom, x_pos, y_pos, z_pos)

; Blitz cylinder mesh is 2x longer than ODE cylinder length
create_Cylinder_Mesh(obj)
ScaleMesh obj\mesh,radius,length/2,radius

RotateEntity
obj\mesh,dGeomGetPitch#(obj\geom),dGeomGetYaw#(obj\geom),dGeomGetRoll#(obj\geom)
PositionEntity
obj\mesh,dGeomGetPositionX#(obj\geom),dGeomGetPositionY#(obj\geom),dGeomGetPositionZ#(obj\geom)

End Function

Function set_material(obj.ODE_object,m.material)
obj\mat\index = m\index
obj\mat\friction = m\friction
obj\mat\density = m\density
obj\mat\restitution = m\restitution
End Function

```

Example Program...

Include "ODE_Object_Library.bb"

```
Include "JV-ode.bb"
Include "ODE_Object_Library.bb"
Graphics3D 800,600,0,0
HidePointer()

Const FPS=60, GRAVITY#=-3.5
Const PERIOD=1000/FPS
Dim objects.ODE_object(200)

ODE_world=dWorldCreate()
ODE_space=dHashSpaceCreate(0)
ODE_contactGroup=dJointGroupCreate(0)

NUM_OBJECTS=50

For n=0 To 200
  objects(n) = New ODE_object
  objects(n)\mass=dMassCreate()
  objects(n)\mat.material = New material
Next; initialise ODE objects array

dWorldSetAutoDisableFlag(ODE_world,1)           ; Turn on auto disable
dWorldSetGravity(ODE_world,0,-2,0)             ; Gravity in m/s/s
dWorldSetERP(ODE_world,0.5)                   ; Global 'error correction'
dWorldSetCFM(ODE_world,0.000001)
dContactSetMode(dContactBounce)
dContactSetBounce(0.2)
dContactSetMu(100.0)

AmbientLight 128,148,168
light1=CreateLight()
TurnEntity light1,45,40,0
LightColor light1,180,160,110
light2=CreateLight()
TurnEntity light2,15,-40,0
LightColor light2,110,160,110

rostrum_pan=CreatePivot()
rostrum_tilt=CreatePivot(rostrum_pan)
camera=CreateCamera(rostrum_tilt)
PointEntity rostrum_pan,camera
PointEntity camera,rostrum_pan
CameraZoom camera,2
PositionEntity camera,0,115,-260
RotateEntity camera,10,0,0

SeedRnd(MilliSecs())

For n=0 To NUM_OBJECTS
  geom_variety=Rnd(1,3)
  material_variety=Rnd(1,3)

  pos_x = Rnd(-40,40)
  pos_y = 100+Rnd(0,100)
  pos_z = Rnd(-40,40)
  siz_x = 3+Rnd(0,5)
  siz_y = 3+Rnd(0,5)
  siz_z = 3+Rnd(0,5)
  rot_x = Rnd(-90,90)
  rot_y = Rnd(-90,90)
```

```

rot z = Rnd(-90.90)

Select material_variety
  Case CON_WOOD
    set_material(objects(n),mat_wood)
  Case CON_GOLD
    set_material(objects(n),mat_gold)
  Case CON_RUBBER
    set_material(objects(n),mat_rubber)
End Select

Select geom_variety
  Case GEOM_CUBE
    create_Box(objects(n),pos_x,pos_y,pos_z,siz_x,siz_y,siz_z,rot_x,rot_y,rot_z)
  Case GEOM_SPHERE
    create_Sphere(objects(n),pos_x,pos_y,pos_z,siz_x,rot_x,rot_y,rot_z)
  Case GEOM_CYLINDER
    create_Cylinder(objects(n),pos_x,pos_y,pos_z,siz_x,siz_y,rot_x,rot_y,rot_z)
End Select
Next

Floor.ODE_Object = New ODE_Object
Floor\geom = dCreatePlane(ODE_space,0,1,0,0)
Floor\mesh = CreatePlane()
EntityColor Floor\mesh,120,120,120

shelf.ODE_Object = New ODE_object
shelf\mat.material = New material
set_material(shelf,mat_wood)
create_Rigid_Box(shelf,20,60,0,180,5,80,0,0,10)

wall.ODE_Object = New ODE_object
wall\mat.material = New material
set_material(wall,mat_wood)
create_Rigid_Box(wall,-140,60,0,10,220,120,90,0,0)
EntityColor wall\mesh,120,20,20

MoveMouse GraphicsWidth()/2,GraphicsHeight()/2
mouse_x#=0
mouse_y#=0
mouse_z#=0

While Not KeyHit(1)

  For ode.ODE_object=Each ODE_object
    If ode\body
      If dBodyIsEnabled(ode\body)
        RotateEntity
ode\mesh,dGeomGetPitch#(ode\geom),dGeomGetYaw#(ode\geom),dGeomGetRoll#(ode\geom)
        PositionEntity
ode\mesh,dGeomGetPositionX#(ode\geom),dGeomGetPositionY#(ode\geom),dGeomGetPositionZ#(ode\geom)
      End If
    End If
  Next

  dSpaceCollide(ODE_space,ODE_world,ODE_contactGroup)
  dWorldQuickStep(ODE_world,0.15)
  dJointGroupEmpty(ODE_contactGroup)

  TurnEntity rostrum_pan,0,mouse_x/5.0,0
  TurnEntity rostrum_tilt,mouse_y/5.0,0,0
  mouse_x=MouseXSpeed()
  mouse_y=MouseYSpeed()
  MoveMouse GraphicsWidth()/2,GraphicsHeight()/2

```

```

mouse z=MouseZ()
mouse_z=mouse_z/10.0
CameraZoom camera,2.0+mouse_z
modifier#=2.0

UpdateWorld
RenderWorld

Flip

Wend

dJointGroupDestroy(ODE_contactGroup)
dSpaceDestroy(ODE_space)
dWorldDestroy(ODE_world)
dCloseODE()

End

```

The above example uses the library functions to populate a scene with static and animated bodies. The library functions are used to add cuboids, spheres and cylinders of a predefined material. The basic structure for adding objects with the ODE_Objects_Library is...

- In you main program include the library file
- Add ODE objects by creating a new instance and create it's material (and mass objects if the object is to be animated as a Rigid Body)

```

object.ODE_object = New ODE _Object
objects\mass=dMassCreate()*
object\mat.material = New material

```

* optional

- Assign the material the object will be made of

```
set_material(object,mat)
```

- Call one of the six object creation functions...
 - Create_Box
 - Create_Sphere
 - Create_Cylinder
 - Create_Rigid_Box
 - Create_Rigid_Sphere
 - Create_Rigid_Cylinder

The Rigid functions produce static objects that are not moved by gravity or collisions. They all take the format of...

```
create_XXX(obj.ODE_object, x_pos#, y_pos#, z_pos#, width#, length#, depth#, x_rot#, y_rot#, z_rot#)
```

Though with differing dimension parameters for boxes, spheres and cylinders.

You can change the look of objects in the Create_XXX_Mesh functions, where you can load in meshes and assign textures for example.