Chapter 11 Activities 2

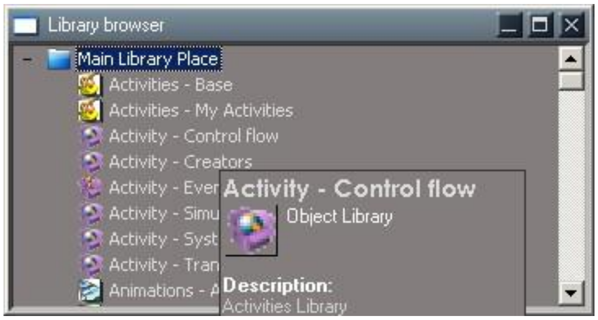
11.1 What is an activity? 2
11.2 Simple Activities 3
11.2.1 Tutorial: Click and Jump 3
11.3 Activity Control Flow 13
11.3.1 Control Flow structures 13
11.3.2 Activity Objects 15
11.3.3 Decision Activity Example 27
11.4 Complex Activities 28
11.4.1 Tutorial: Secure Fence 28
11.4.2 PacMan 31
11.5 Physics Activities 36
11.5.1 Tutorial: Wind Tunnel 39
11.5.2 Tutorial: The Wheel 46
11.5.3 Tutorial: AirHockey Game 53

Chapter 11 Activities

11.1 What is an activity?

An **Activity** is a sequence of actions with a duration in **time.** Unlike animations (even procedural animations), an activity has a **goal**, which the activity participants try to accomplish. In order to accomplish the goal, user must be able to **interact** with objects or other users during the activity. An activity can be a task like painting a texture, playing a simple game, or a collaborative modeling effort with other trueSpace users.

Activities are the building blocks for trueSpace-based applications, and they are accessible to you for creating your own applications, even without programming a single line of code. The trueSpace **Link Editor** provides flexible editing tools for creating and modifying activities in a convenient way.



Activity object libraries.

With activity control flow tools you can create and modify behaviors in either small or large contexts, such as game rules, physical laws (physics), and manipulation access. You can also create behaviors associated with specific objects (models) like driving a vehicle, walking an avatar, or an engine simulation. Also, the **Link Editor** will assist you with the construction of compound and scripting **commands**. Combining behaviors with objects is a powerful new way to create complex procedural, interactive behaviors. It empowers the objects in trueSpace; it gives them life.

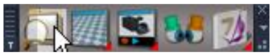
11.2 Simple Activities

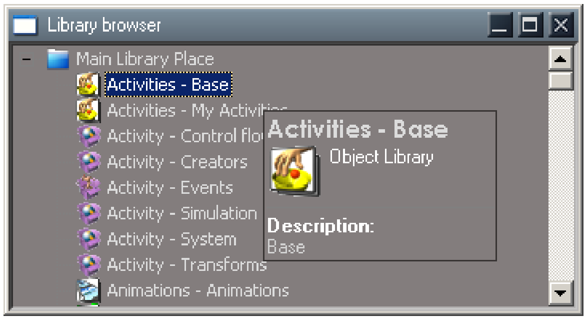
11.2.1 Tutorial: Click and Jump

In this tutorial you will create a simple scene with several objects, then begin an activity called Click and Jump. The Click and Jump activity will cause the selected object to perform a flip when you click on it.

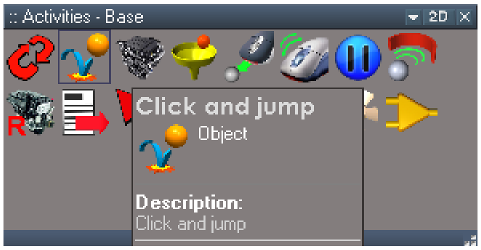
Step 1: Start trueSpace in a new scene configuration, or if trueSpace is running, select New Scene from your toolbar.
Step 2: From your "**Activities/Base**" Library, drag and drop the "Click and jump" library object into the space 3D context of your **Link Editor** window.

If your library is not showing open the library browser and then open the library in the stack from here by either a double click or using the r-click menu.

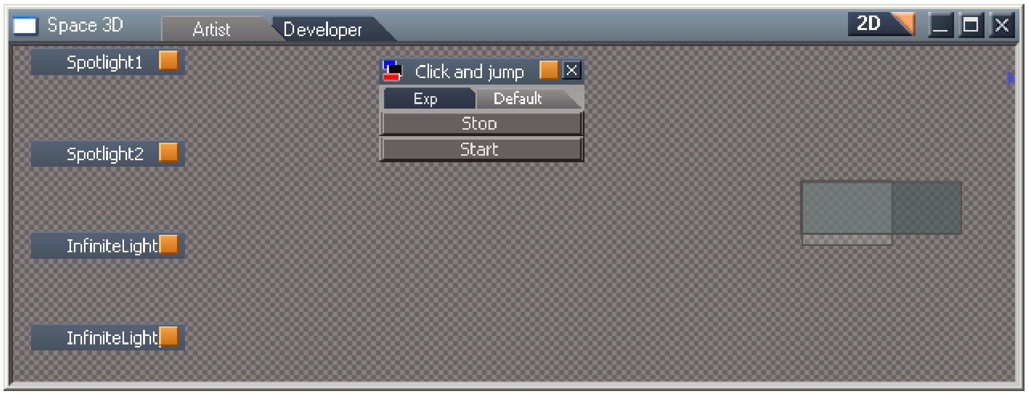

Library Browser Icon



Activities Base Library



Click and jump object



Click and jump object in Link Editor.

Step 3: From your **Objects/Base** Library, drag the Rhino and Dino items into the **Workspace** , you can add some primitives from the toolbar as well , here I added some text and a sphere .

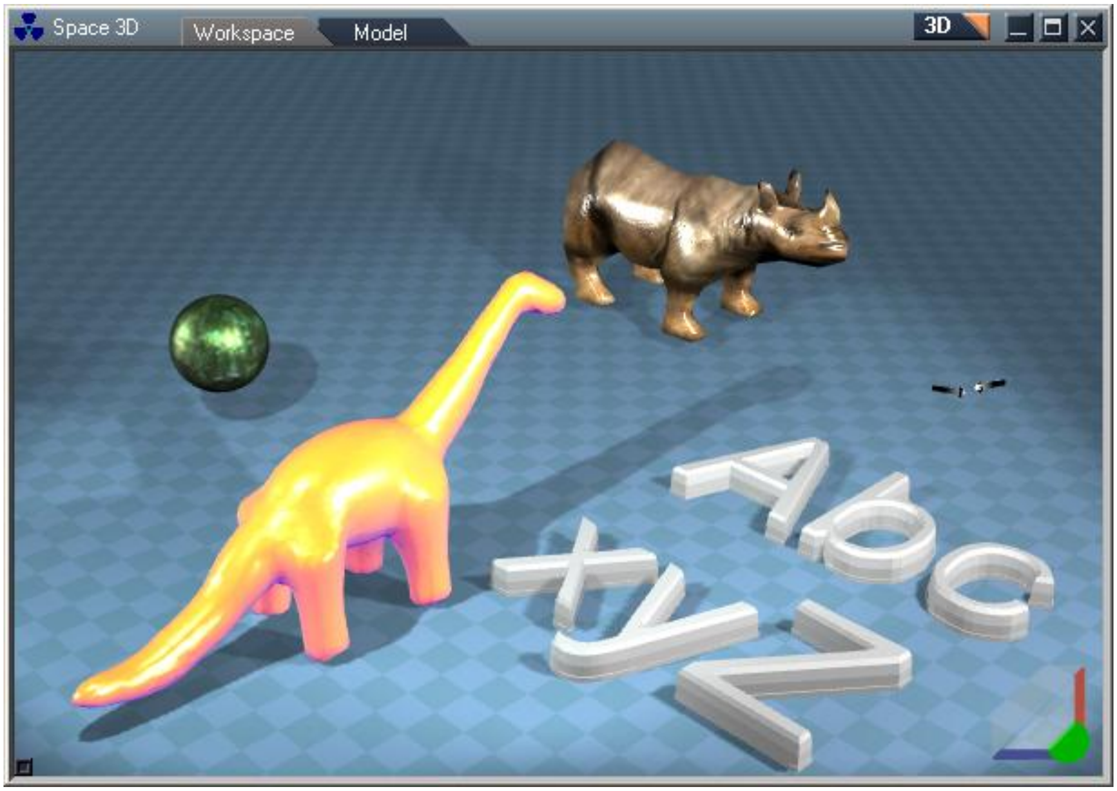


Figure 2: Add some objects to Space 3D.

Step 4: Start the activity by pressing the Click and Jump's "Start" button. Select the Sphere, and then select the Dino.

As you select an object, it will “jump” once. You must select a new object before the activity will cause the object to jump.

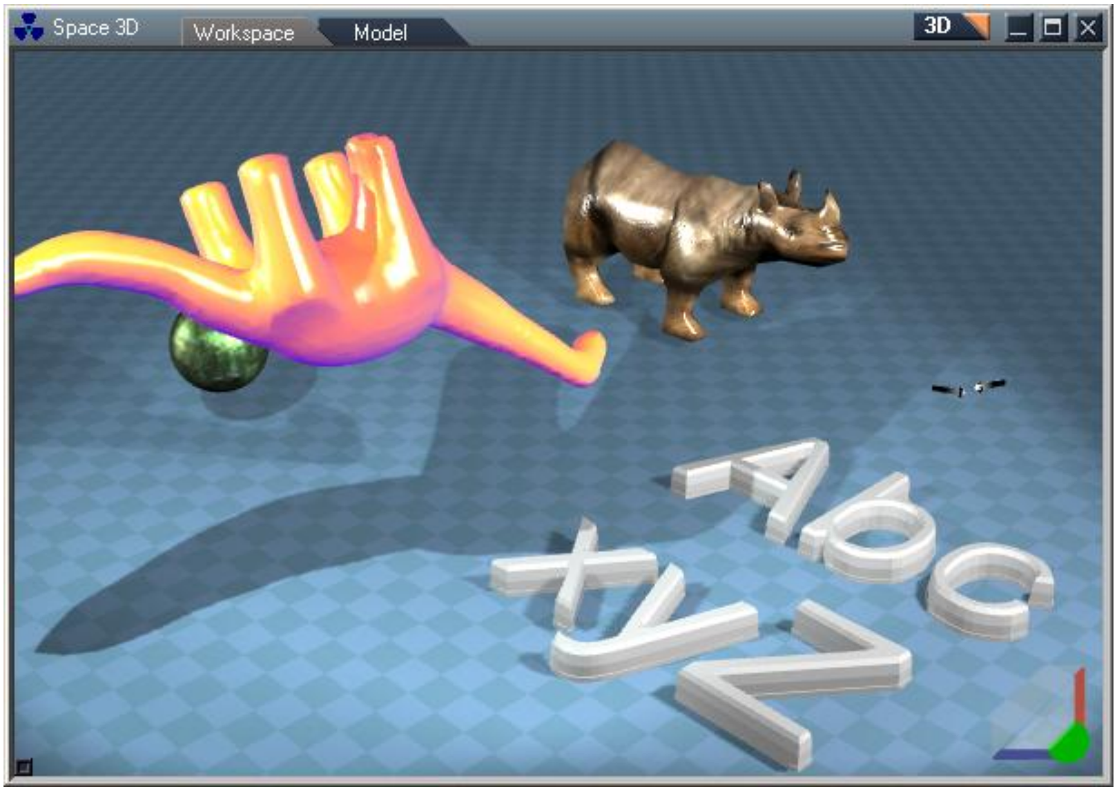
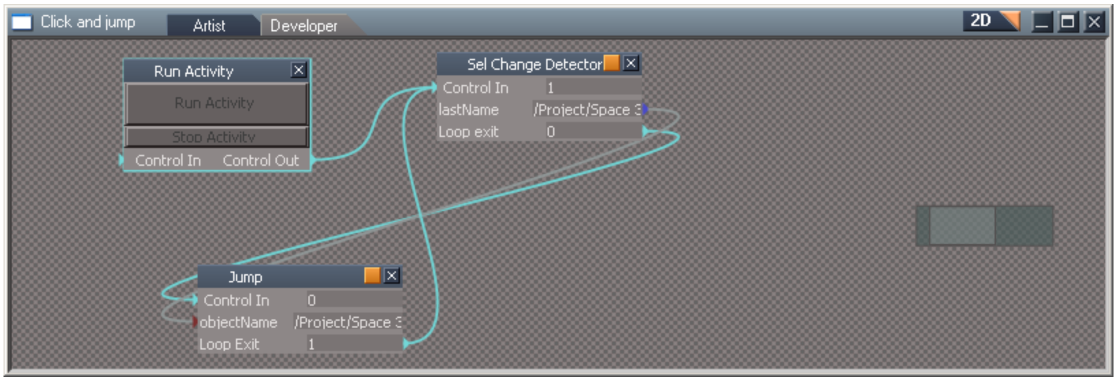


Figure 3: Click and jump in action.

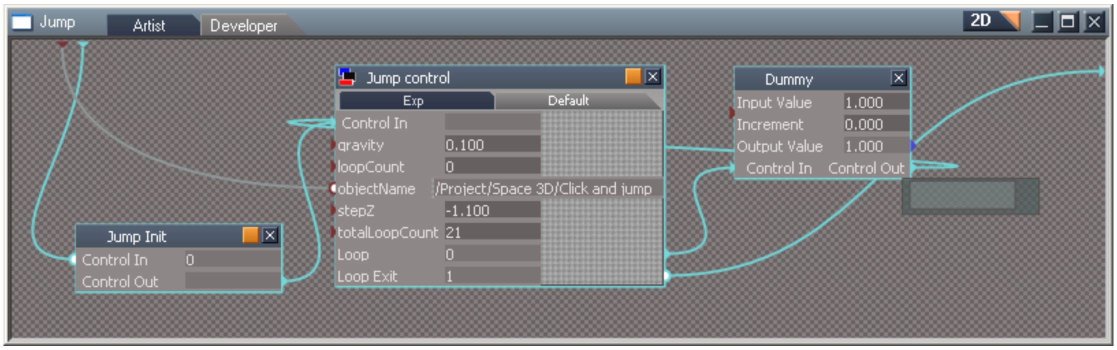
The image above simulates the Dino object’s jump. You have created a simple activity, but it can lead to other, more complex activities.

Step 5: Once you are finished experimenting with the activity, press Click and Jump’s “Stop” button.

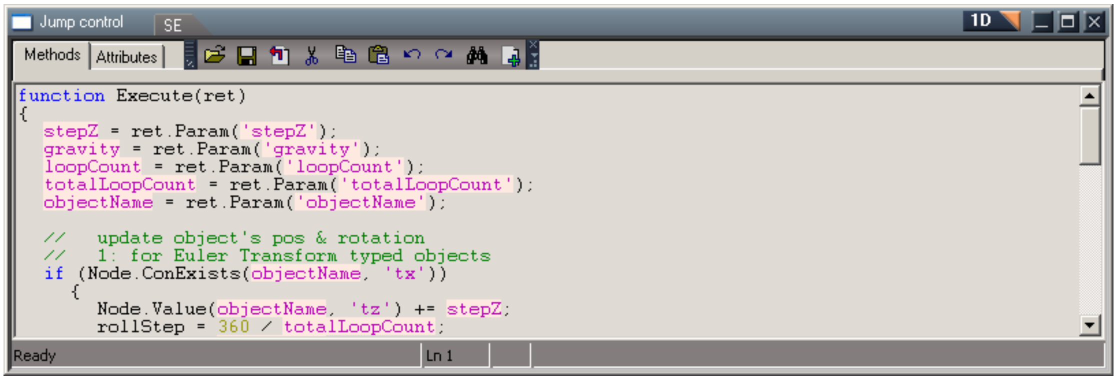
Step 6: From Link editor, click the orange square on the Click and Jump object. This will open the object and transport you and your view inside it. Once inside, you can see that the Click and Jump object is composed from other objects which are linked together into some sort of a **Structure.** This structure is what gives an object its essence. Programmers call this structure a **Method (or Function)** and generally use all kind of esoteric tools like VisualStudio.NET to create and edit it, but the Link Editor allows you to create and edit this structure without actually programming although it still allows for you include a script of your own or re-use an existing one to take part in your own activities.



Click and jump object, entered in Link Editor.



Jump object, entered in Link Editor.



```
function Execute(ret)
{
   stepZ = ret.Param('stepZ');
   gravity = ret.Param('gravity');
   loopCount = ret.Param('loopCount');
   totalLoopCount = ret.Param('totalLoopCount');
   objectName = ret.Param('objectName');

   //   update object's pos & rotation
   //   1: for Euler Transform typed objects
   if (Node.ConExists(objectName, 'tx'))
      {
        Node.Value(objectName, 'tz') += stepZ;
        rollStep = 360 / totalLoopCount;
```

Jump control script entered in Link Editor.

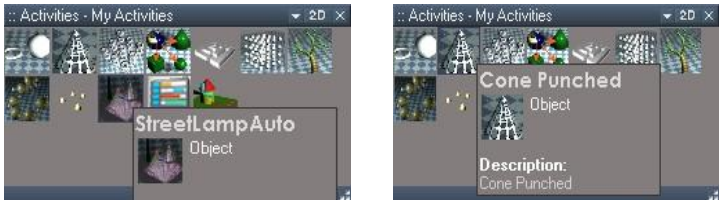
Use the orange triangle near the 1D in the title-bar to exit the Script Editor

11.2.2 Tutorial: Street Lamp - Sensor

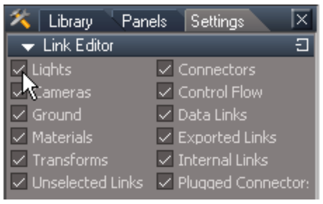
In this tutorial you will create a simple scene with a smart Street Light, which automatically turns on when the selected object moves closer to the center of the base.

Step 1: Start a **New Scene** with the **Workspace** view visible.

- Load the **Activities/My Activities** library using the Library Browser, drag and drop the **StreetLampAuto** activity into the **Workspace** or **Link Editor** view.
- Drag the **Cone Punched** object into the **Workspace** or **Link Editor** view.



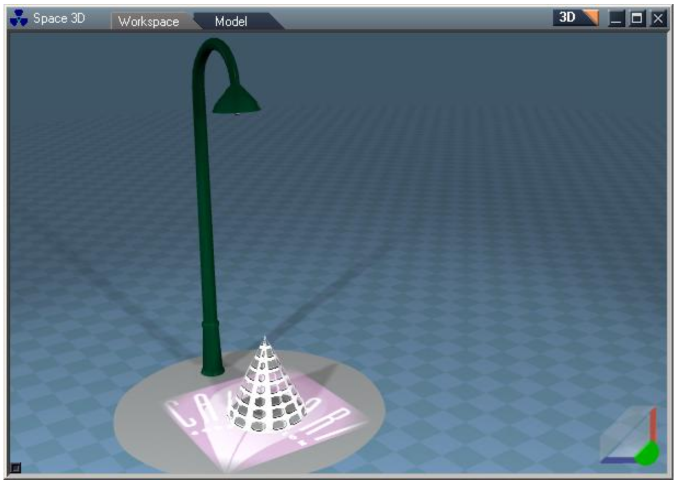
StreetLampAuto activity example objects.



✓ Lights	✓ Connectors
✓ Cameras	✓ Control Flow
✓ Ground	✓ Data Links
✓ Materials	✓ Exported Links
✓ Transforms	✓ Internal Links
✓ Unselected Links	✓ Plugged Connector:

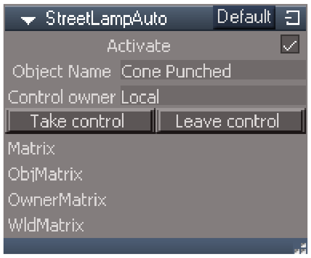
Link Editor Settings.

Make sure "Lights" is enabled in the Link Editor section of the Workspace Settings.



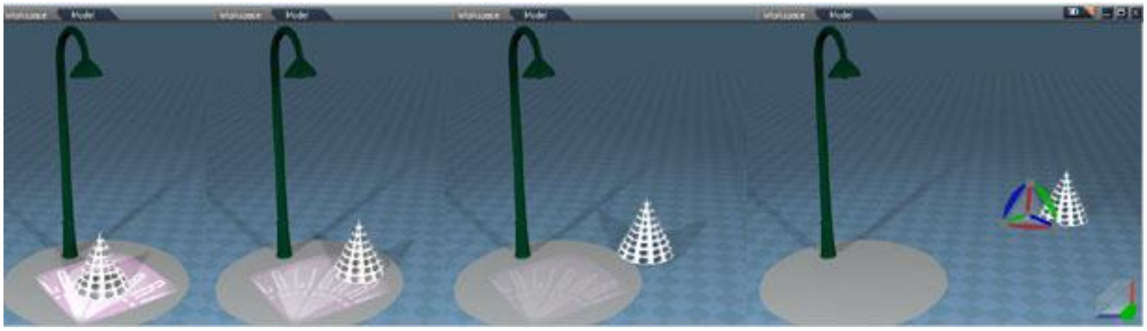
StreetLampAuto and Cone Punched object in the scene.

Step 2: In the **Link Editor**, activate the **StreetLampAuto** object by left-clicking in the check-box. This tells the object to start calculating how close the selected object is to the center of the circle-base under the Street Lamp. As the cone object is moved closer or further away, the intensity of this lamp increases or decreases accordingly.

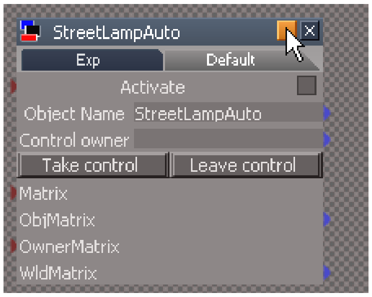

StreetLampAuto Default
Activate ✓
Object Name Cone Punched
Control owner Local
Take control | Leave control
Matrix
ObjMatrix
OwnerMatrix
WldMatrix

Activate the StreetLampAuto.

Step 3: Once you have activated the **StreetLampAuto** object, select the Cone object and move it around the Street Lamp. You will notice that as the Cone draws nearer to the center of the Street Lamp's base, the lamp brightens. Move the Cone away from the Street Lamp, and the light dims.

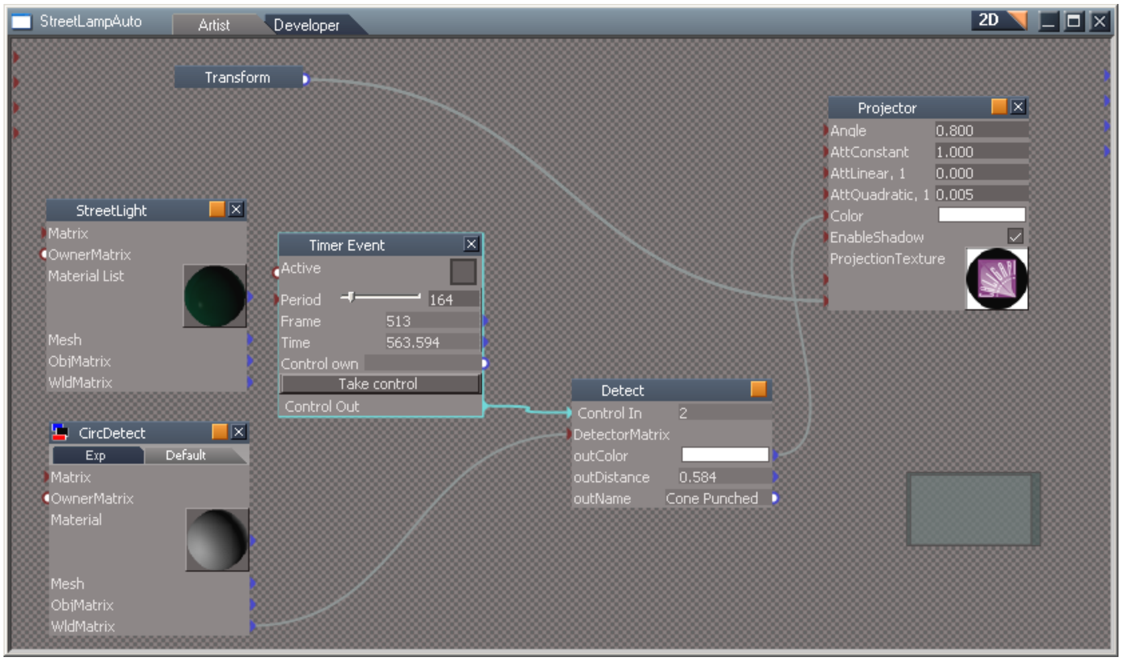

StreetLampAuto in action

Next we'll enter the StreetLampAuto object in the LE using the orange square and look at the various building blocks used to enable the scenario.
If you cannot see an orange square you may need to choose the default aspect for the object.


StreetLampAuto
Exp | Default
Activate
Object Name StreetLampAuto
Control owner
Take control | Leave control
Matrix
ObjMatrix
OwnerMatrix
WldMatrix

StreetLampAuto in default Aspect

Once you are inside you may want to arrange the objects to see them all more clearly as shown in the following image. You can switch the LE to either Developer or to Artist Tab to give a quick clean up of the connectors and links that show to allow for a neater presentation of the objects and give a less cluttered feel to the LE without having to use the filters in the preferences for the LE controls.



Projector: Angle 0.800; AttConstant 1.000; AttLinear, 1 0.000; AttQuadratic, 1 0.005; Color; EnableShadow ✓; ProjectionTexture

StreetLight: Matrix; OwnerMatrix; Material List; Mesh; ObjMatrix; WldMatrix

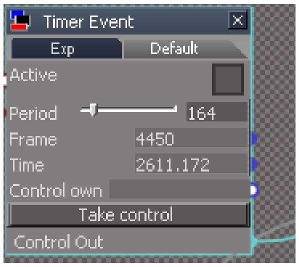
Timer Event: Active; Period 164; Frame 513; Time 563.594; Control own; Control Out

Detect: Control In 2; DetectorMatrix; outColor; outDistance 0.584; outName Cone Punched

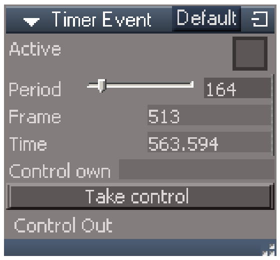
CircDetect: Matrix; OwnerMatrix; Material; Mesh; ObjMatrix; WldMatrix

Inside view of StreetLampAuto object in Artist tab.

Here is a breakdown of the various sub-objects within the **StreetLampAuto** object:



Active; Period 164; Frame 4450; Time 2611.172; Control own; Control Out



Active; Period 164; Frame 513; Time 563.594; Control own; Control Out

LE View Timer Event Stack View.

Timer Event: This object is the engine that drives the StreetLampAuto object. Once activated, this object sends a control signal to the Detect object. The Timer Event Period Slider adjusts how often the activity will check for changes in position. The Period is measured in milliseconds (1/1000). The higher the number for Period, the slower the reaction of the activity to changes in position of the selected object.

- Activate - Activates or deactivates the timer. This can be performed by anyone who has privileges to modify the timer.
- Period – Defines the period in milliseconds when the “tick” will be generated.

The first tick is generated after the first period and not immediately after the timer is started.

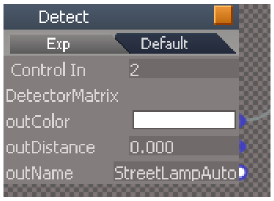
- Frame – The current frame of the timer is always an integer.
- Time – Time in seconds for how long the timer is running.

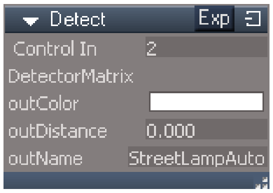
  This value together with the Frame value is not reset when you disconnect from shared space or if you just stop the timer. To reset the timer it is necessary to reset one of these connectors.
- Control own is used only in shared space.

  This is a string value that represents the computer on which the activity is running. As a default it is running server. In other cases it will list the name of the participant who took control of this timer. If the field is empty, then the activity is executed on your own computer while you are not connected in shared space or the timer is stopped.
- Take control – If you click on this button and you are logged into a shared space, then the activity will be executed on your computer instead of on the server or another computer. This setting is useful for example, if the activity depends on your own settings. eg: The Street Lamp Auto object uses your current selection and will work properly only if you take control.
- Leave control – Click this button to leave control.

  If you are in shared space the then activity will run on the server
- Control Out – Calls your activity each the timer ticks.

Note: The Time connector stay synchronized with the server's time, but the ratio between the Time and Frame connectors will be different after you disconnect from shared space.

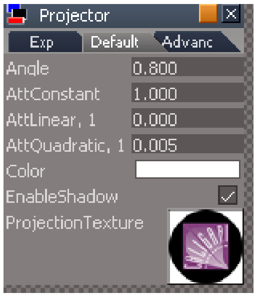

Detect
Exp | Default
Control In 2
DetectorMatrix
outColor
outDistance 0.000
outName StreetLampAuto


Detect | Exp
Control In 2
DetectorMatrix
outColor
outDistance 0.000
outName StreetLampAuto

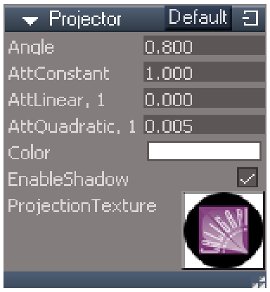
LE view Detect object Stack view.

Detect:

This object serves as the "brain" of the smart lamp. Some methods of Detect object detect the center of the CircDetect object, which of course is the circle polygon we see at the base of the Street Lamp. The CircDetect's WldMatrix output connector sends the information across the wire to the Detect object. Other methods capture the name of the currently selected object (Cone Punched in this example) and call the Cone object's matrix information to find out where its center is located. The distance between the center of the circle and the center of the selected object is calculated. There are also some triggers involved that determine the brightness (color) of the lamp. In this particular example, there is a trigger at distance of 7. If the distance is greater than 7, the lamp is black or dimmed. Once the selected object moves closer, the lamp light changes from black to dark gray, to white, as the selected object approaches circle's center.



Angle 0.800
AttConstant 1.000
AttLinear, 1 0.000
AttQuadratic, 1 0.005
Color
EnableShadow ✓
ProjectionTexture

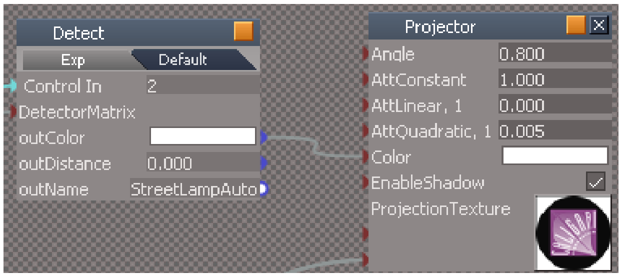


Angle 0.800
AttConstant 1.000
AttLinear, 1 0.000
AttQuadratic, 1 0.005
Color
EnableShadow ✓
ProjectionTexture

LE Projector object Stack.

Projector:

The Projector object is the actual light in trueSpace that is being adjusted as the selected object moves closer or further away from the circle's center. In this example, the Detect object is setting a value for the color of the light. This value is passed from the Detect's outColor connector to the Projector's Color input connector.



Control In 2
DetectorMatrix
outColor
outDistance 0.000
outName StreetLampAuto

Angle 0.800
AttConstant 1.000
AttLinear, 1 0.000
AttQuadratic, 1 0.005
Color
EnableShadow ✓
ProjectionTexture

Detect's outColor connected to the Projector's Color input connector.

Step 4: Now that you have an idea as to the logic behind this activity and how the activity itself will function, it is time bring a few extra **StreetLampAuto** objects into our scene. From the **Activities/My Activities** library, drag some additional **StreetLampAuto** objects into the **Link Editor** window.

- Note that **StreetLampAuto** objects are actually occupying the same space in 3D (check your **Workspace window** once you have the Street Lamps loaded into the scene).
- Once you have dragged the objects into the **Link Editor**, move them apart in the **Workspace window**.

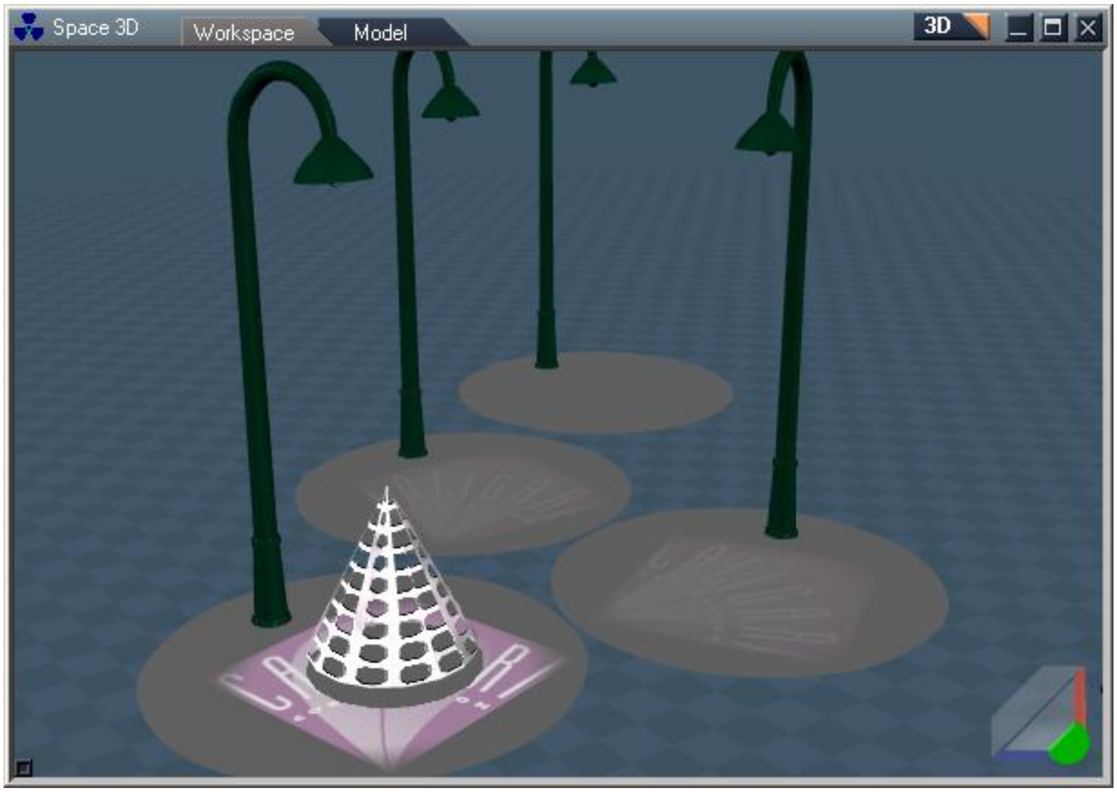


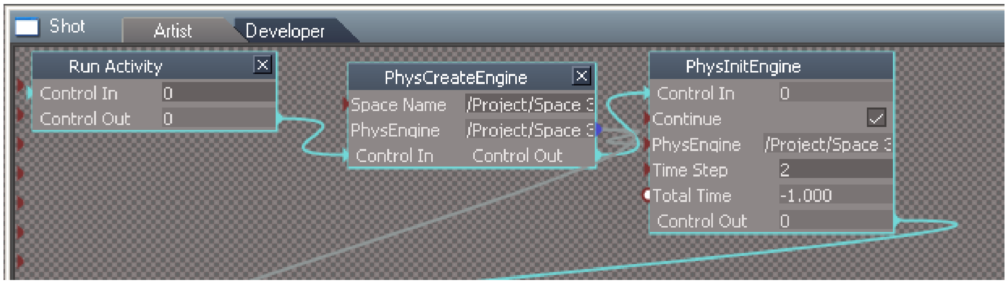
Figure 6: Spread out the Street Lamps.

Step 5: Once you have activated and spread out the Street Lamps, select the Cone object and drag it around the scene. You should see that each lamp lights up as Cone object moves closer to it.

11.3 Activity Control Flow

11.3.1 Control Flow structures

Control flow objects allow you to explicitly decide what actions (commands) are executed at a particular **time** and in which order. They include blue control connectors with blue links. Unlike gray links, which represent a flow of data, no data flows along blue links, just a control signal determining which action will be have control next. In the image below, once the **RunActivity** Event object sends a control signal to **PhysCreateEngine**, it loses control and **PhysCreateEngine** acquires it and then passes it onto the **PhysInitEngine,** it loses control and passes the control message to the next linked object in the activity, until it reaches the end of the line and then stops when the conclusion or decision is reached .



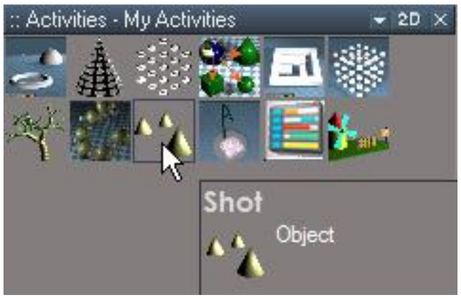
Run Activity
Control In 0
Control Out 0

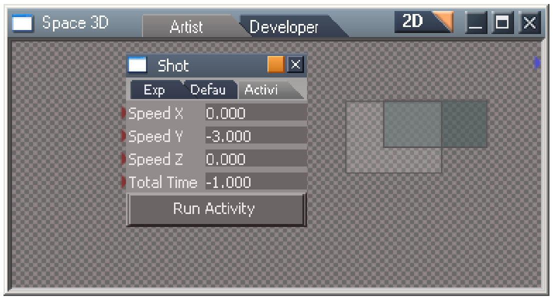
PhysCreateEngine
Space Name /Project/Space 3
PhysEngine /Project/Space 3
Control In Control Out

PhysInitEngine
Control In 0
Continue ✓
PhysEngine /Project/Space 3
Time Step 2
Total Time -1.000
Control Out 0

Inside the Shot object the control flow passes on to each object in the linked chain to reach its decision

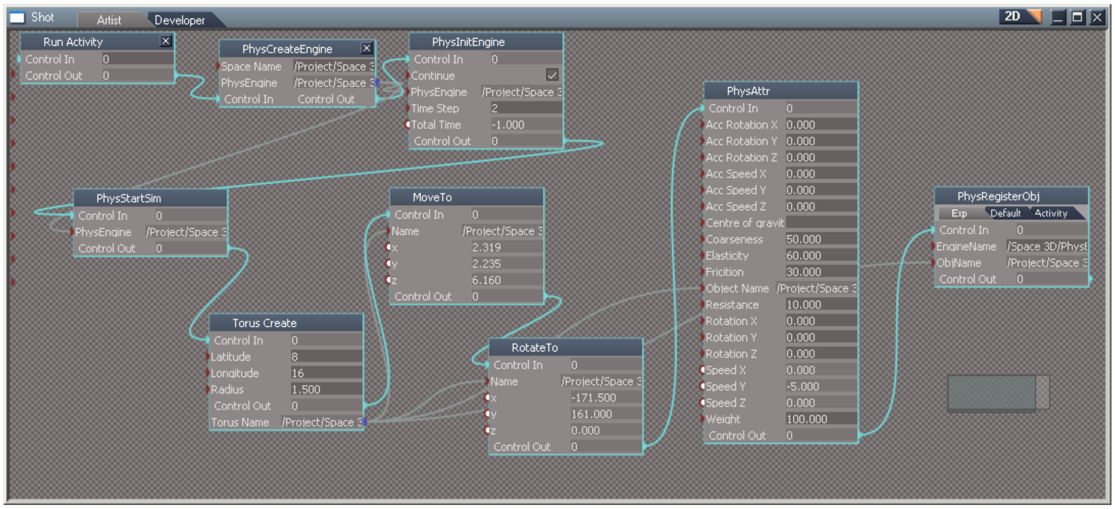
Connecting control objects resembles visual programming, complete with switches, loops, etc. Complex activities can be assembled quickly simply by using blue links to create activity control flow. Control links and data links can be combined within one activity, even within one object, as you can see from the Shot activity example:





Speed X 0.000
Speed Y -3.000
Speed Z 0.000
Total Time -1.000

Shot Activity object ***Shot Activity object dragged to LE***



Run Activity
Control In 0
Control Out 0

PhysCreateEngine
Space Name /Project/Space 3
PhysEngine /Project/Space 3
Control In Control Out

PhysInitEngine
Control In 0
Continue ✓
PhysEngine /Project/Space 3
Time Step 2
Total Time -1.000
Control Out 0

PhysStartSim
Control In 0
PhysEngine /Project/Space 3
Control Out 0

MoveTo
Control In 0
Name /Project/Space 3
x 2.319
y 2.235
z 6.160
Control Out 0

Torus Create
Control In 0
Latitude 8
Longitude 16
Radius 1.500
Control Out 0
Torus Name /Project/Space 3

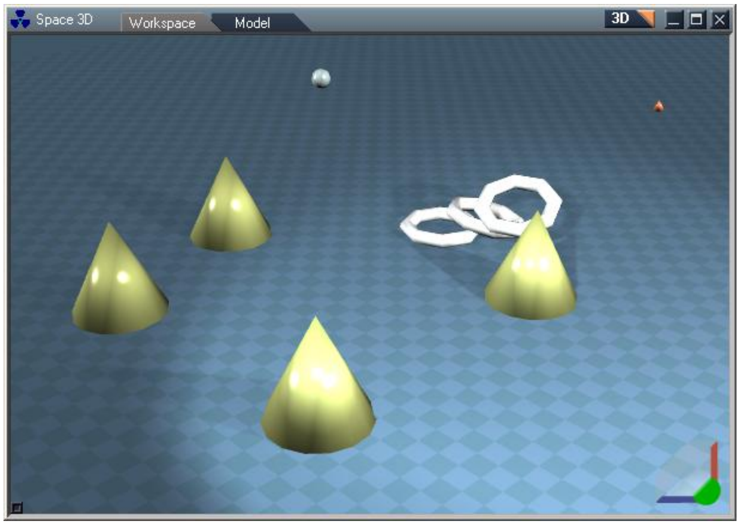
RotateTo
Control In 0
Name /Project/Space 3
x -171.500
y 161.000
z 0.000
Control Out 0

PhysAttr
Control In 0
Acc Rotation X 0.000
Acc Rotation Y 0.000
Acc Rotation Z 0.000
Acc Speed X 0.000
Acc Speed Y 0.000
Acc Speed Z 0.000
Centre of gravit
Coarseness 50.000
Elasticity 60.000
Fricition 30.000
Object Name /Project/Space 3
Resistance 10.000
Rotation X 0.000
Rotation Y 0.000
Rotation Z 0.000
Speed X 0.000
Speed Y -5.000
Speed Z 0.000
Weight 100.000
Control Out 0

PhysRegisterObj
Control In 0
EngineName /Space 3D/PhysE
ObjName /Project/Space 3
Control Out 0

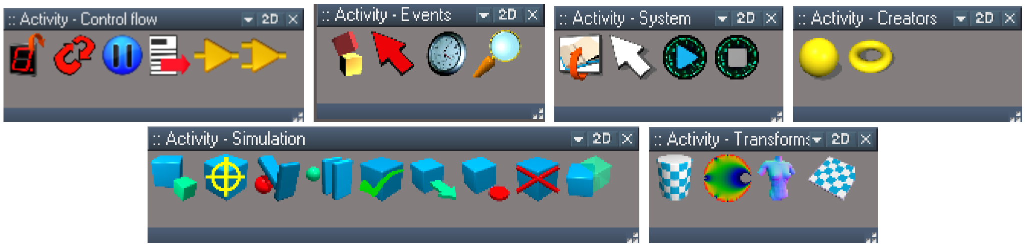
Shot Activity blue control links

In fact, control links are not strictly necessary, and any behavior could be created by using data links only. The benefit of control links is that **they make time an explicit part of trueSpace activities,** and time is of paramount importance in making trueSpace objects **live.**



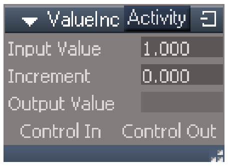
Shot Activity in action

11.3.2 Activity Objects



Activity object libraries

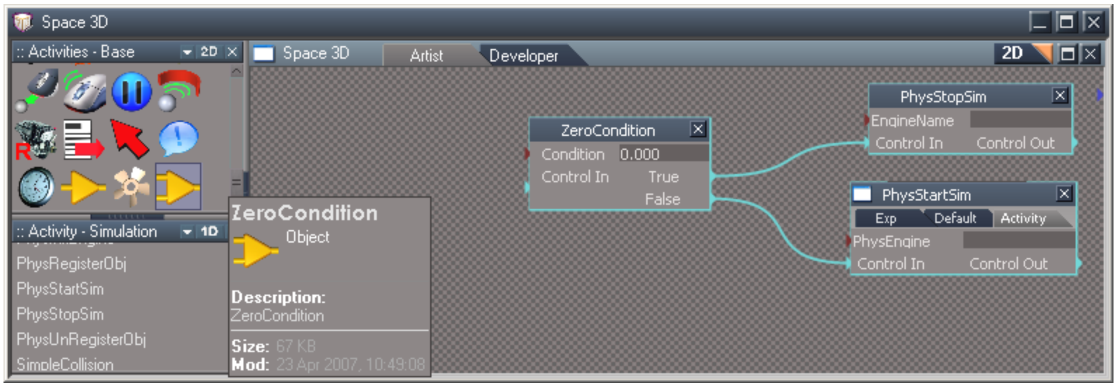
A simple *activity* object represents an action or sequence of actions. This activity object usually has one *input* connection from the previous object and one *output* connection indicating the transition to the next activity object. A simple activity object can have several other input and output attributes. The data supplied by the *input* attributes can be used in the execution of the action, and the results are stored in the *output* attributes to be used (sent as messages) in other objects.


ValueInc Activity
Input Value 1.000
Increment 0.000
Output Value
Control In Control Out

A simple activity object

Decision Activity Object

A *Decision* activity object represents an action where a decision is necessary. This activity object has one *input* connection and usually at least two *output* connections indicating the next activity objects. A *Guard* condition is used to evaluate transitions in the flow.



ZeroCondition
Object
Description:
ZeroCondition
Size: 67 KB
Mod: 23 Apr 2007, 10:49:08

ZeroCondition
Condition 0.000
Control In True
False

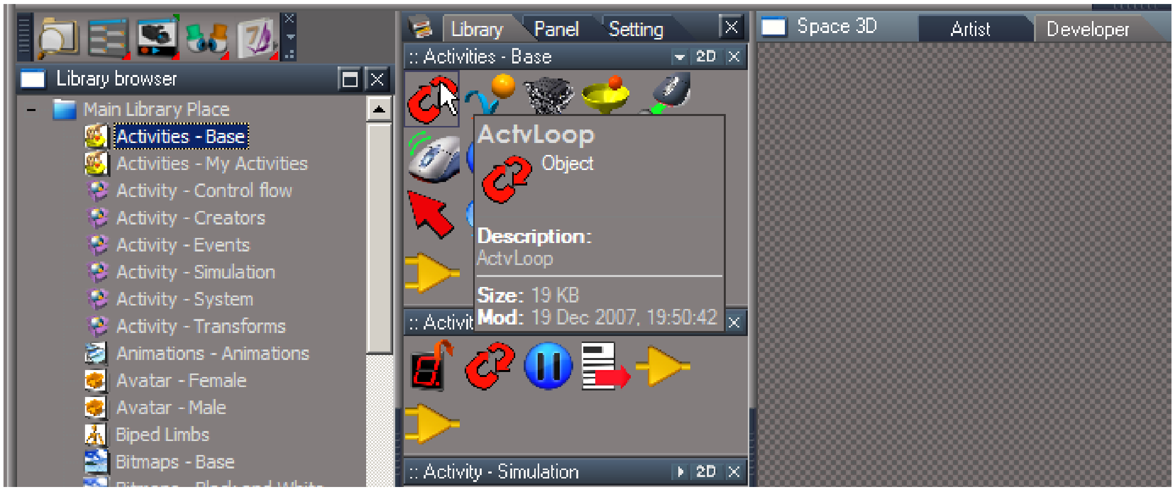
PhysStopSim
EngineName
Control In Control Out

PhysStartSim
Exp Default Activity
PhysEngine
Control In Control Out

ZeroCondition object

Loop Activity Object

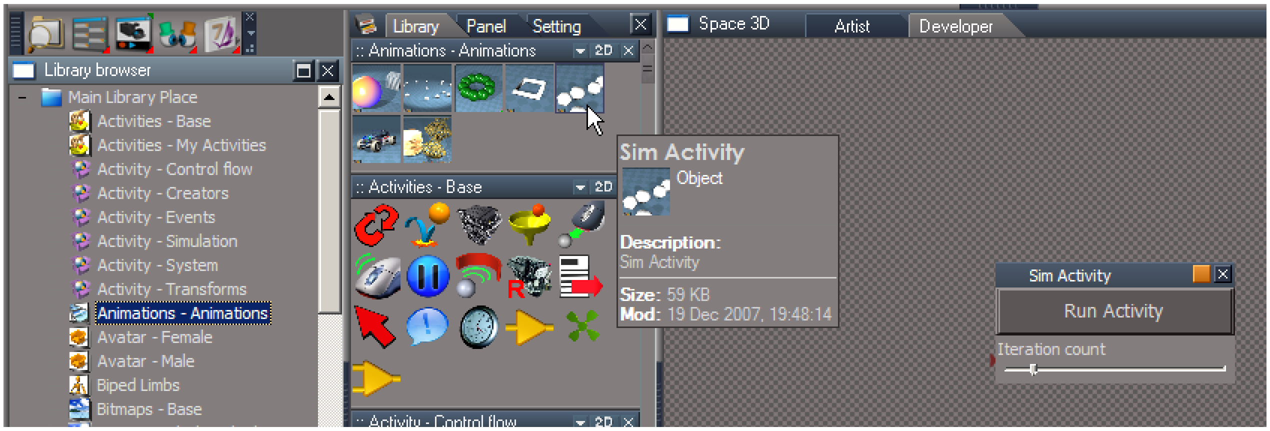
Allows an action to be executed a specified number of times.



ActvLoop
Object
Description:
ActvLoop
Size: 19 KB
Mod: 19 Dec 2007, 19:50:42

ActvLoop object

Example object: The Sim Activity in the Animations - Animations library gives an example of its use .



Sim Activity
Object
Description:
Sim Activity
Size: 59 KB
Mod: 19 Dec 2007, 19:48:14

Sim Activity shows an example of the ActvLoop usage

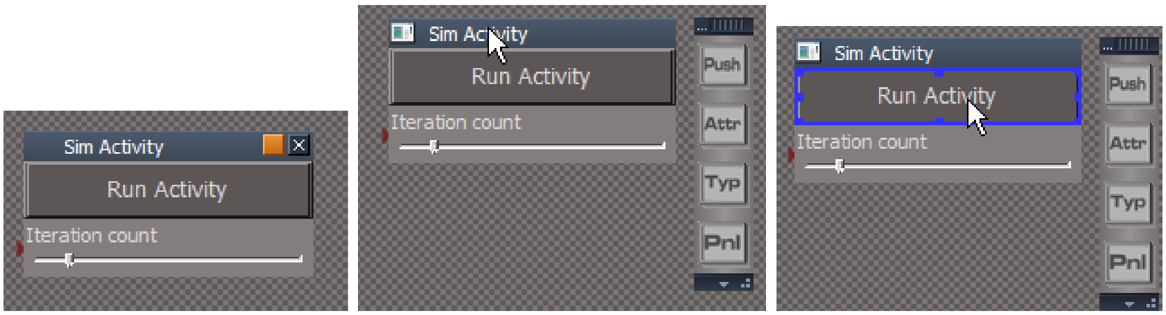
The iteration count on the front of the panel controls the amount of looping. This is a connector that been exported from the ActvLoop activity encapsulated within the Sim Activity object so that the values can be changed without having to enter the object again.

The Run Activity button fires a command to run the first activity in the chain of linked behaviors inside the object. To see how this command is run:

1: Right-click the title bar of the object this will put you into a Panel Edit Mode, this is an advanced mode that allows for interfaces to be defined and created through the use of context specific toolbars it is covered in more detail in its own section of the manual Chapter 2 User interface.

2: select the button

3: Right-click over the button control to show the properties .

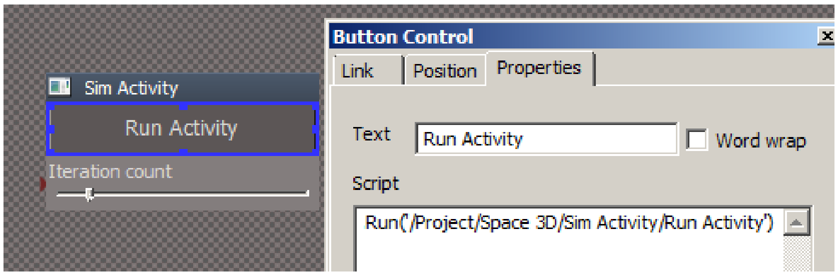


Enter Panel edit Mode to see the buttons script properties in the Sim Activity;
1:R-click title bar. 2: Select button.

In here you will see this text in the script dialog window: *Activity.Run('/Project/Space 3D/Sim Activity/Run Activity')*

Basically this in effect tells the first activity *('/ inside the /project/scene/object/name of activity: to run')*, which in turn passes the control to the other activities linked together in the chain.

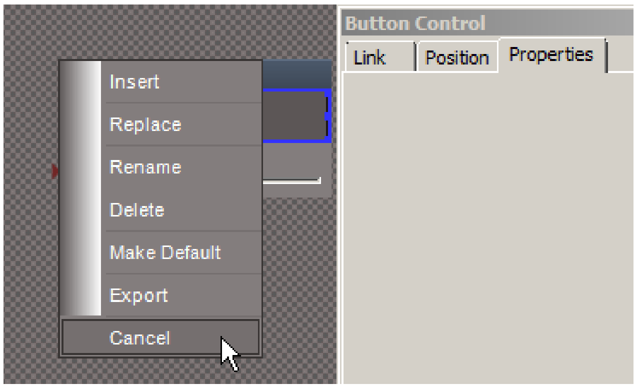
You could add other buttons in Panel Editing Mode which would set different activities or chains of activities within the same object so using these methods alternative behaviors' could run depending which buttons were pressed .



Text: Run Activity
Word wrap: ☐
Script: Run('/Project/Space 3D/Sim Activity/Run Activity')

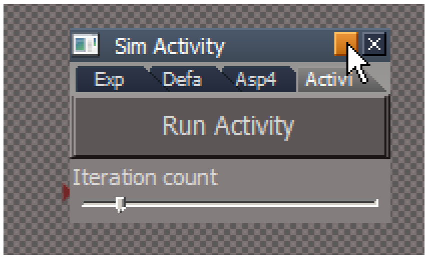
3: R-click to show the buttons properties.

After looking at the script for the button you can exit the objects Panel Edit mode by a R-click in the Sim Activity title bar and choose cancel from the menu.



4: R-click in the title bar and choose cancel.

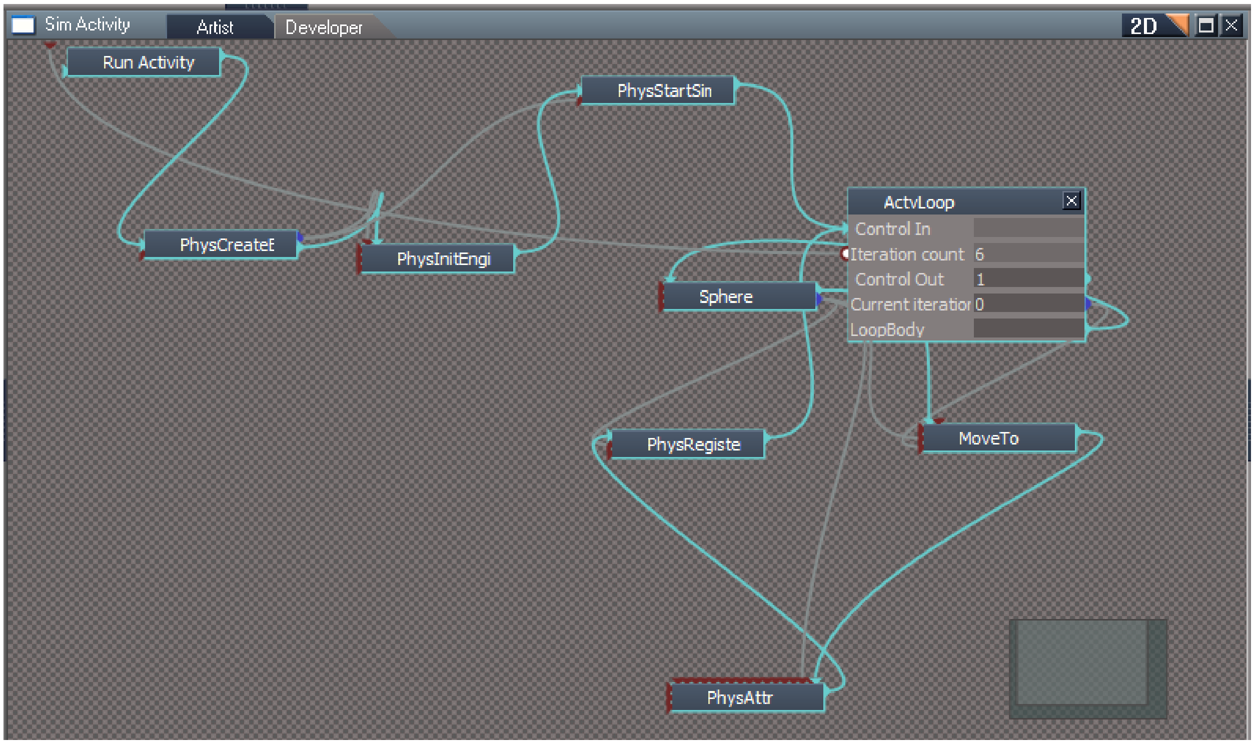
Next we can take a look inside the Sim Activity and then you can study how its linked together more closely: Enter the Sim Activity object by a left-click on the orange square in its title bar.



Enter the object by the orange square.

Once inside you can see the control flow passing from one object to the other represented by the blue wires and can see the ActvLoop and see how its linked to form the loop body , also note the exported Iteration count which is visible on the top level of the object.

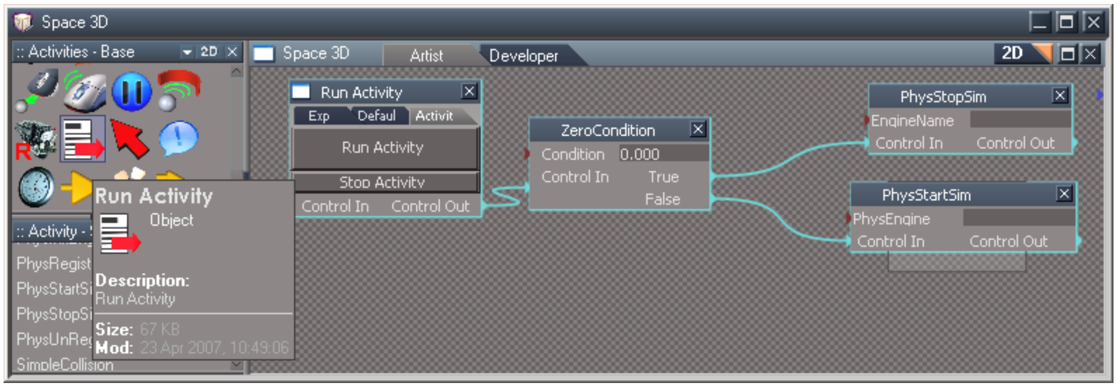
When you've finished looking at how the control passes through the activity to achieve its goal and how the loop body is formed, Navigate back out of the object to scene level by using a left-click on the orange triangle in the LE title bar.



The control flow and the ActvLoop object inside the Sim Activity object

Run and Stop Activity

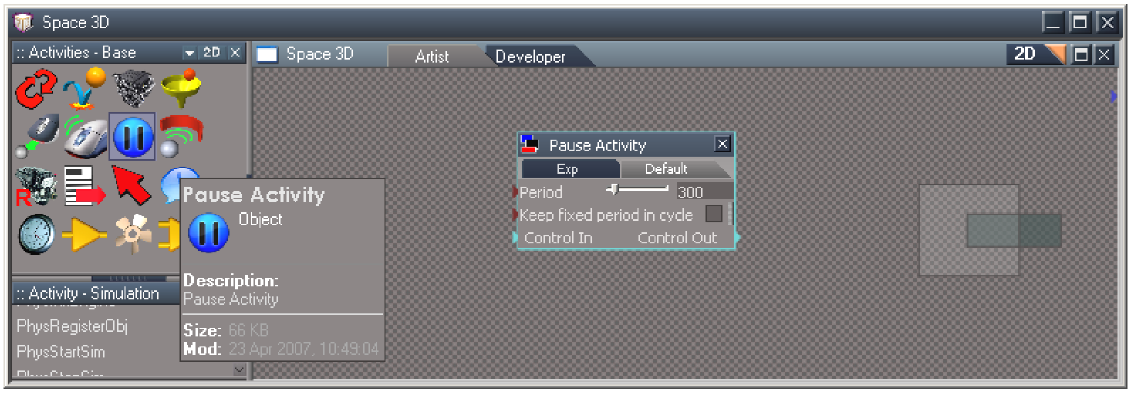
Allows you to start and stop an activity by clicking a button. Note: Most activity objects placed in libraries have their own start/stop buttons in their panels, so there is no need to use the universal Run Activity starter for them.



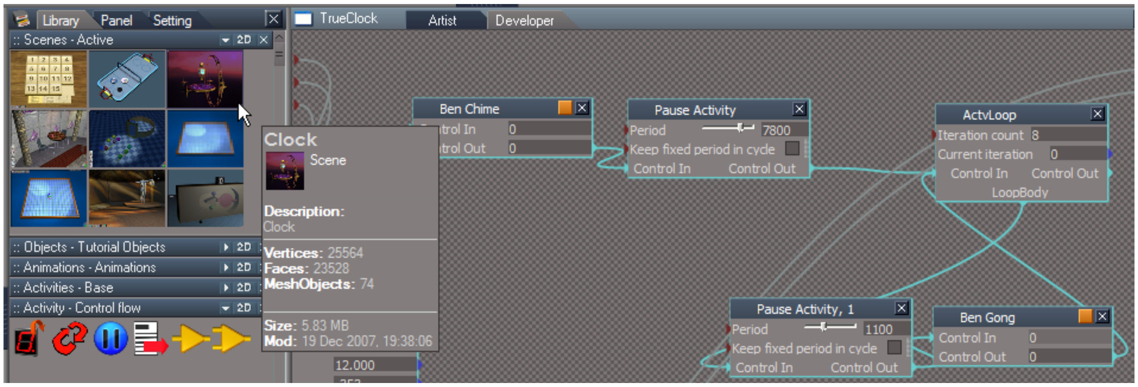
Run Activity object

Pause Activity Object

Allows you to control the speed of activity loops, keep a constant speed of loop execution on various machines, and save some CPU time (to not drastically slow down other processes in trueSpace, like scene navigation, while the activity is running). It is strongly recommended that you put the Pause Activity with loop-based activities. Examples of use can be seen by looking at the Clock scene which is in the Scenes -Active library and navigating into the TrueClock object.



Pause Activity object

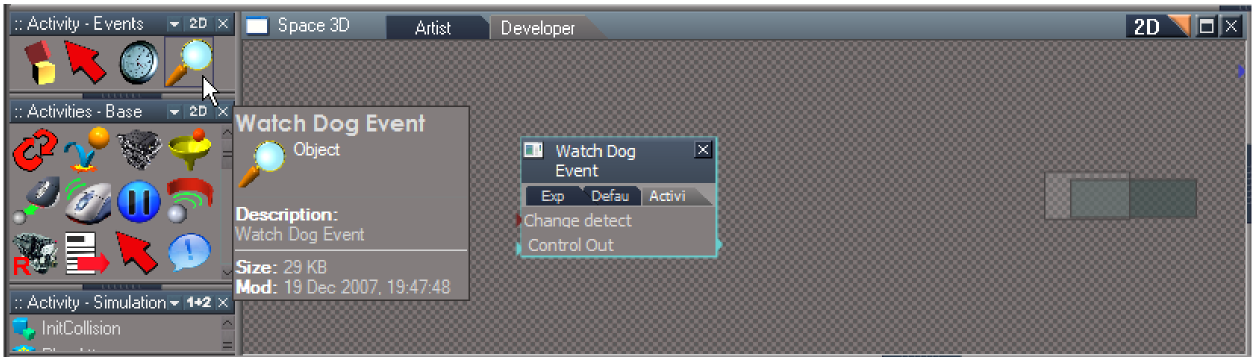


Pause Activity object inside the TrueClock

Event activities

You can use special event activities to trigger your activities in different ways. The advantages to using events are saving CPU time, and the ability to detect particular changes / situations in the scene in an elegant way. The types of events are as follows:

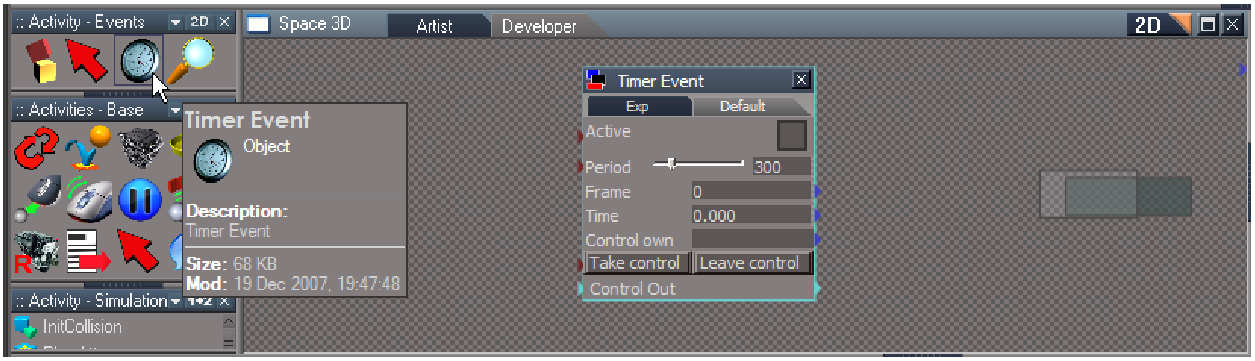
- Watch dog event: Fires the activity when an output attribute change is detected. (For example, you can attach it to a matrix attribute of the object to perform a special action on the object's movement or rotation.) Refer to the Scene Deformer demo in the Activities library, to get a better idea on how it works.



Watch Dog Event
Object
Description:
Watch Dog Event
Size: 29 KB
Mod: 19 Dec 2007, 19:47:48

Watch Dog Event object

- Timer event: Fires the activity each period of time (default is 300 ms). Similar to the Timer object but contains control out link and the Activate checkbox to start / stop. For an example, refer to the Living circle demo in the Animation library.



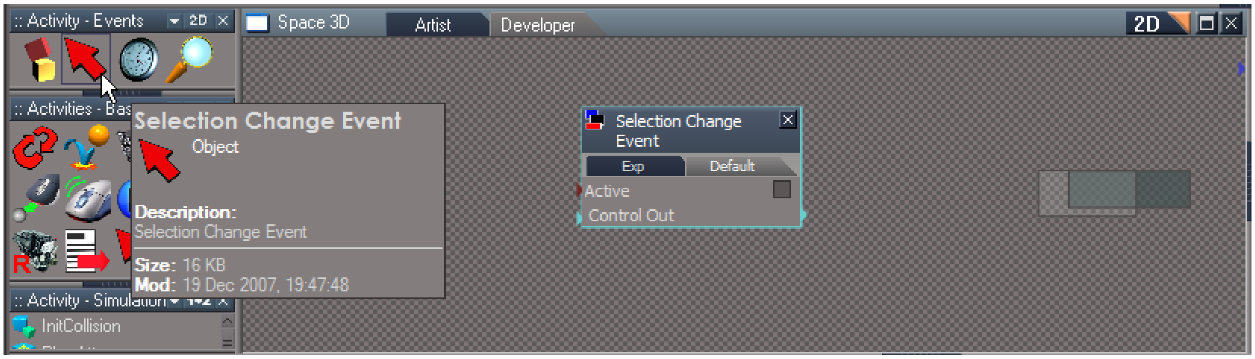
Timer Event
Object
Description:
Timer Event
Size: 68 KB
Mod: 19 Dec 2007, 19:47:48

Active
Period 300
Frame 0
Time 0.000
Control own

Timer Event object

- Selection change event:

Fires the activity each time the object selection has been changed (e.g. you click on a different object in the LE or D3D view). Refer to the Scene Deformer demo in the Activities library for an example.



Selection Change Event
Object
Description:
Selection Change Event
Size: 16 KB
Mod: 19 Dec 2007, 19:47:48

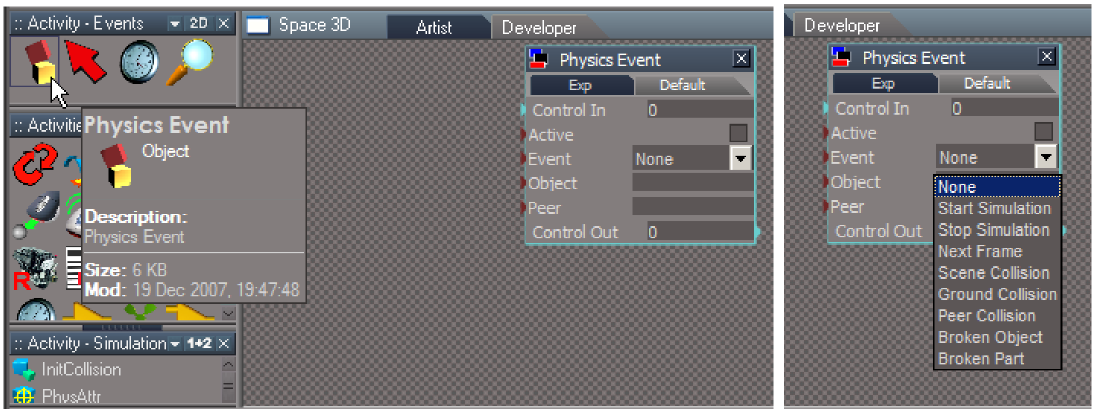
Selection Change Event
Active
Control Out

Selection Change Event object

- Physics Event:

Physics Event enables to receive and monitor special events coming from a running simulation. When you start a simulation in Workspace, this object will listen to user defined events in the simulation and will fire immediately when the defined event occurs in the scene. The Physics Event activity can be used with scripts or activities and gives an opportunity to respond to events in user defined way.

The Physics Event has two controlling connectors for connecting to other activities and four input connectors, that allow to activate and set events.



Physics Event
Object
Description:
Physics Event
Size: 6 KB
Mod: 19 Dec 2007, 19:47:48

Physics Event
Control In 0
Active
Event None
Object
Peer
Control Out 0

Physics Event object

- Active activates and deactivates Physics Event node, node will receive information from simulation only when Active is checked.

For the Physics Event to be considered during physics you need to put a check in the Active connector on the Physics Event, before starting physics. When the object is active the Physics Event monitors the objects named Object and Peer linked to the inputs by a collision event defined by the dropdown menu ;
When a collision is detected an activity or script can be made to run.

- Event sets the type of event to be monitored. There are nine values that can be chosen for this attribute:

The choices available via a drop down menu to trigger and generate an event are as follows:

- None: No event is monitored, event will not fire even though it is in active state
- Start Simulation: Event is generated when simulation is started. Example of use refer to 3D SoundBall
- Stop Simulation: Event is generated when simulation is stopped. Example of use refer to 3D SoundBall
- Next Frame: Event is generated after simulation frame was saved to keyframes (Note: the option Save Anim on phys engine panel has to be checked to receive this event)

Next five events are for the monitoring of collision events for objects included in a simulation:

- Scene Collision: Event is generated when object with the name that is stored in Object connector collides with any objects in the scene
- Ground Collision: Event is generated when object with the name that is stored in Object connector collides with ground
- Peer Collision: Event is generated when object with the name that is stored in Object connector collides with object with name that is stored in Peer connector.
- Broken Object : when controlled object is breaking.
- Broken Part : when controlled object is broken from parent object.

- Object contains the name of object for which events as Scene Collision, Ground Collision and Peer Collision should be monitored, the object has to have physical attributes assigned, and it has to be included in simulation to receive given events (either as active object or physical object with animation with unchecked Active attribute). Sub-object of physical objects inlcuded to the simulation can be also set to this connector, in this case only collisions for given sub-object will be monitored (for example when you have character object, you can set to this connector the name of bone for which you would like to monitor collision events).

- Peer contains the name of peer object that is used when *Peer Collision* is set. This object can be any objects from Workspace.

In workspace, you can create and use more Physics Event nodes to monitor one or more objects. For one physical object in simulation, more events are allowed to be detected using more Event nodes (for example different kind of events can be monitored for different parts of physics object). When simulation is not running, the events are not generated and node is not working.

Example scenes which can be unzipped imported and studied:

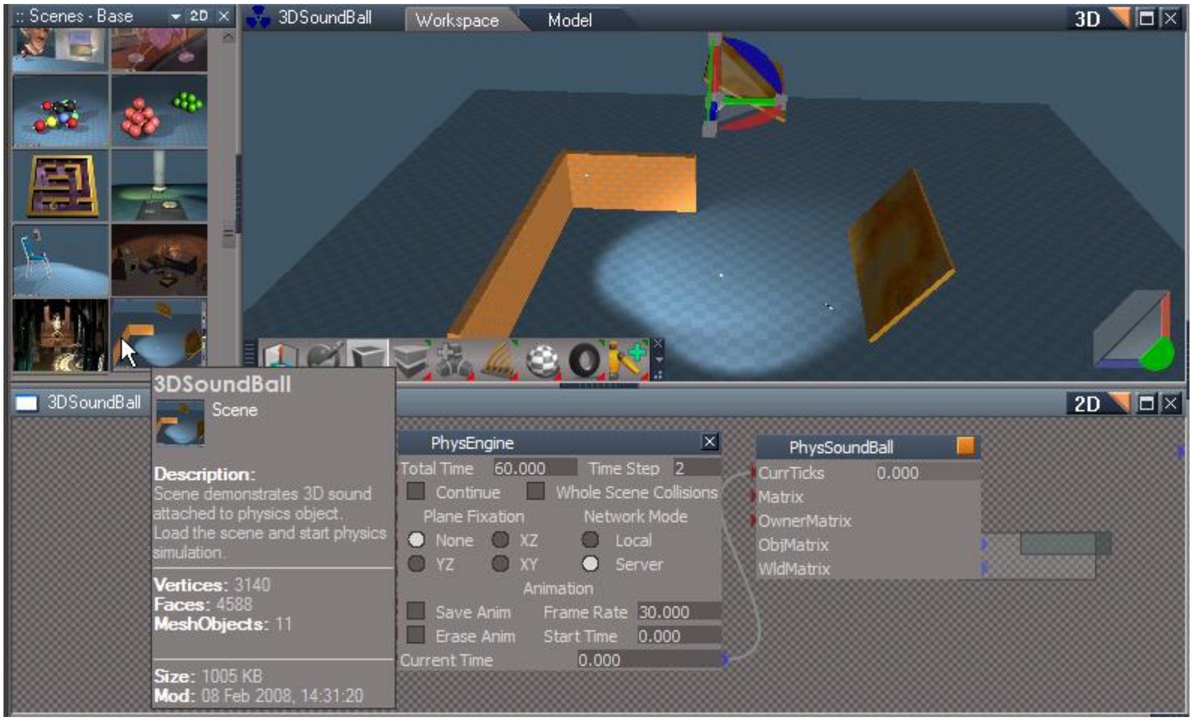
- Physics Event 1 monitors the collision between the orange cubes, when a collision is detected, then a simple script that opens a system Alert window is shown.

- Physics Event 2 shows a character with physics and animation and car moving towards the character, the event monitors for a collision between them and when the collision is detected, the animation for the character is turned off and physics is turned on using simple scripts.
- Physics Event 3 this scene is very similar to second one, but the event node only monitors a collision between the characters head and one cube, when the collision is detected physics is turned on for the character by a simple script.

Example scene : Scenes-Base library:

- 3D SoundBall demonstrates how you can attach sound to physics events using the Link Editor.

To see how the events are triggered and used navigate into the PhysSoundBall, then look inside the PhysSound



3DSoundBall
Scene

Description: Scene demonstrates 3D sound attached to physics object. Load the scene and start physics simulation.

Vertices: 3140
Faces: 4588
MeshObjects: 11

Size: 1005 KB
Mod: 08 Feb 2008, 14:31:20

PhysEngine
Total Time 60.000 Time Step 2
Continue
Whole Scene Collisions
Plane Fixation: [✓ None] XZ YZ XY
Network Mode: Local [✓ Server]
Animation
Save Anim
Erase Anim
Frame Rate 30.000
Start Time 0.000
Current Time 0.000

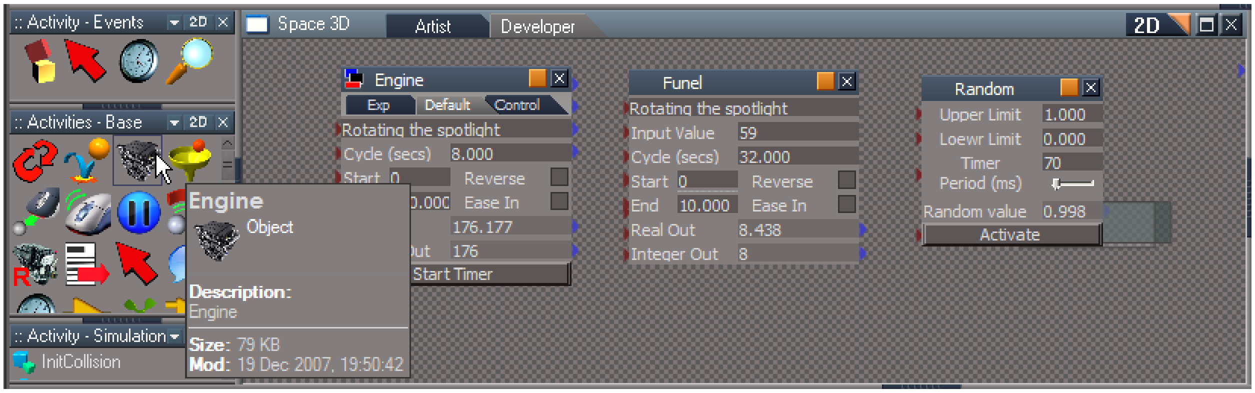
PhysSoundBall
CurrTicks 0.000
Matrix
OwnerMatrix
ObjMatrix
WldMatrix

3D SoundBall Scene

Engines

Engine and Funnel do not have control links. Engine is a more robust timer which will allow you to precisely control types and timing of various real time activities. Funnel is useful for precise real time control using time varying attributes of other objects.

The Random engine, found in the library next to the previous engines, produces random output values in given range (the Upper Limit and the Lower Limit attributes) and in the given speed (the Timer period attribute).



The engine objects

Script access to activities

By using the following command you can start the processing of any particular activity:

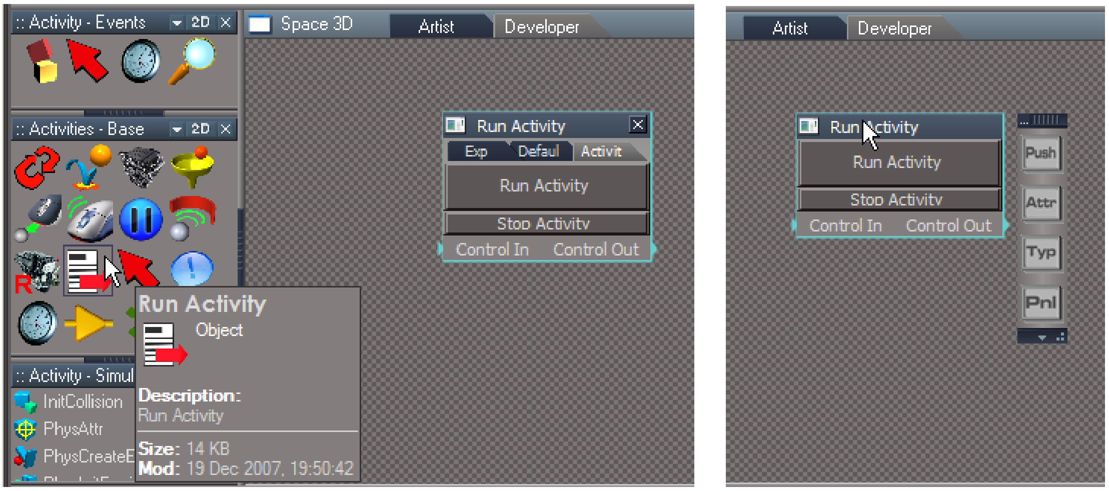
Activity.Run('<full name of object>').

The object identified by "full name" must have at least one control input attribute.

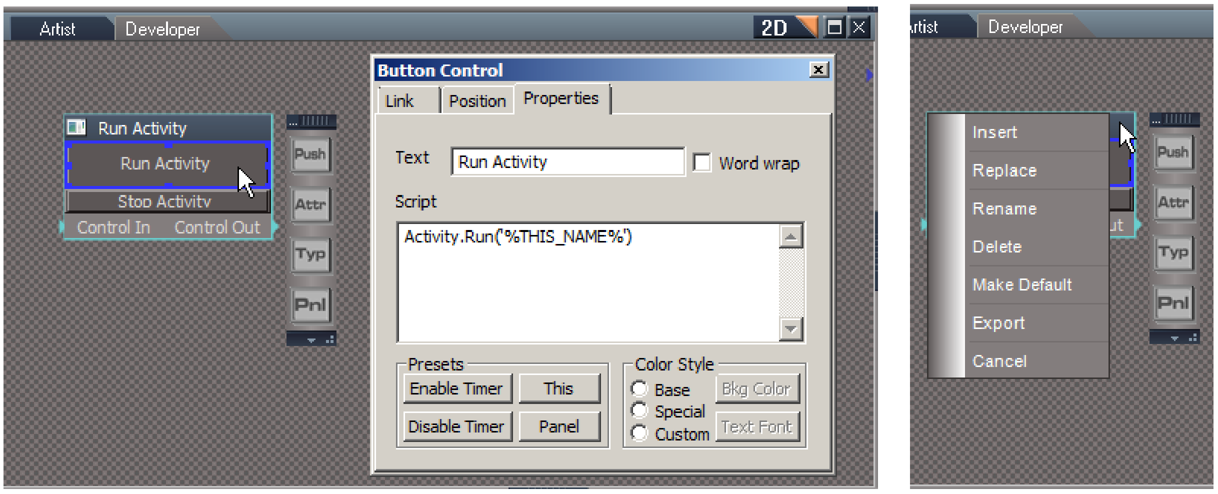
The **Activity** library contains the *Run Activity* object. It defines a button with the run activity preset:

Activity.Run('/Space 3D/Run Activity').

Clicking the button starts the activity identified by the object.



Run Activity object

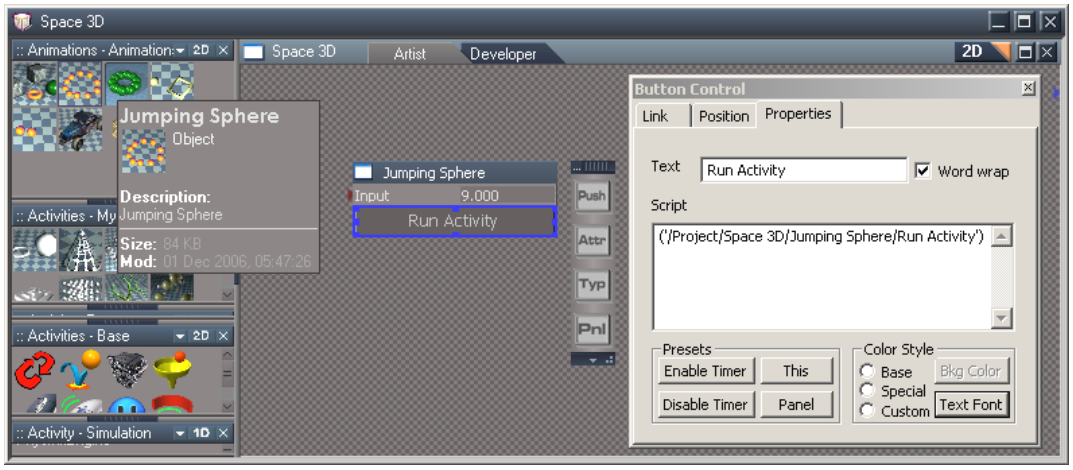


Text: Run Activity
Word wrap: ☐
Script:

```
Activity.Run('%THIS_NAME%')
```

Run Activity Preset

There is an example in the Animations Library in the jumping sphere, the script in the properties of the button that calls the activity that's inside the object is typed like this *Activity.Run('/Project/Space 3D/Jumping Sphere/Run Activity')*

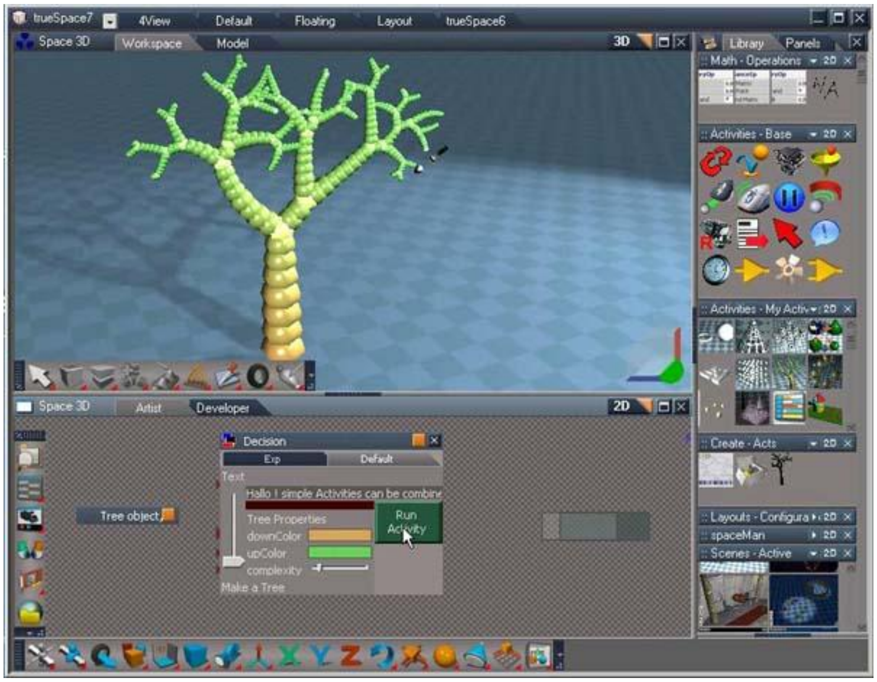


Text: Run Activity
Word wrap: ☑
Script:

```
('/Project/Space 3D/Jumping Sphere/Run Activity')
```

Run Activity by Name of activity

11.3.3 Decision Activity Example



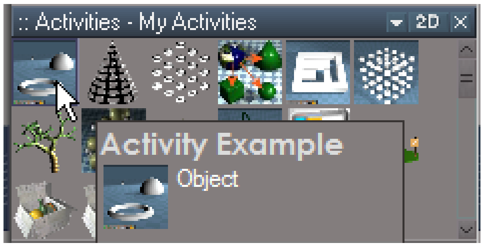
Decision Activity

video link

A positive or negative number passed by the slider, determines which chain of activity will run. In this example the RunActivity object inside the Decision object was simply re-named to Make a Choice as the name for the activity to be run when the button is pressed.

The True or False values when passed determines which other activity branch will Run , whether it makes a tree or decides to speak.

Another example is the Activity Example in the my activities library.



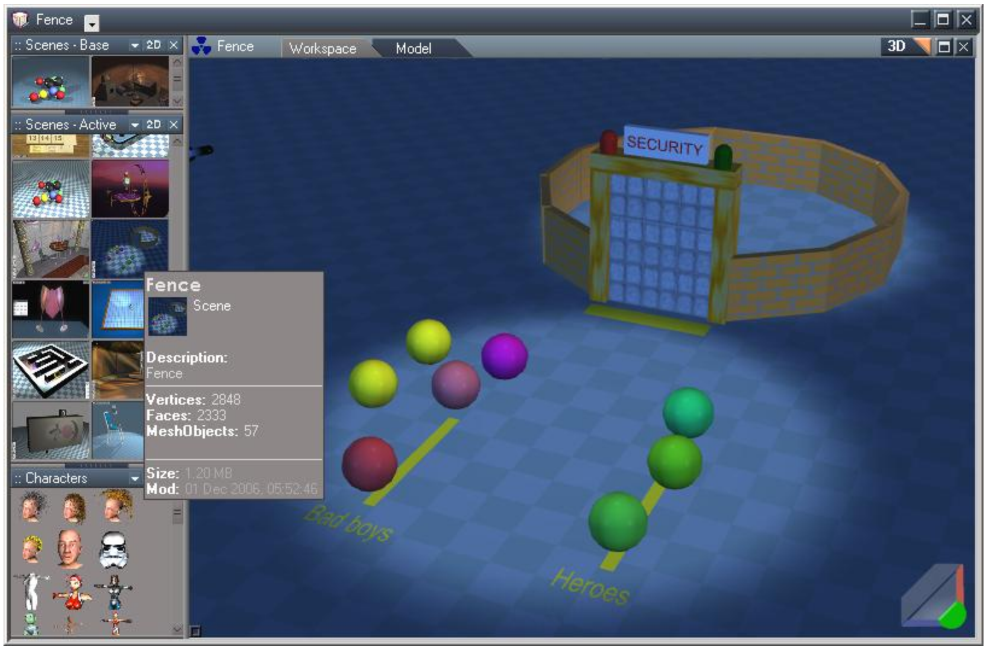
11.4 Complex Activities

11.4.1 Tutorial: Secure Fence

In this tutorial you will investigate a more complex activity titled **Secure Fence**. It uses an **Optical Detector** to determine the color of a sphere and its proximity to the secured door. The interesting thing about this activity is that it is built entirely within Link Editor by linking objects, and there is no scripting involved at all!

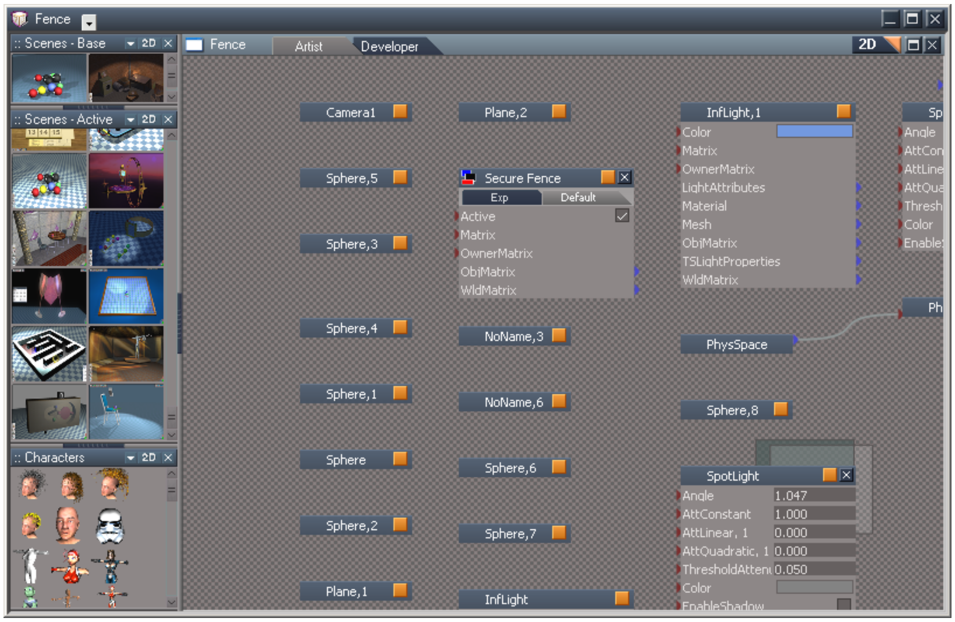
Step 1: Load the Secure Fence scene by left-clicking on it in the Scenes-Active library. Once in the scene, you will notice the following:

- Green Spheres
- Non-Green Spheres
- Secured Door and room, with an Optical Detector (located at the foot of the door).



Fence
Scene
Description: Fence
Vertices: 2848
Faces: 2333
MeshObjects: 57
Size: 1.20 MB
Mod: 01 Dec 2006, 05:52:46

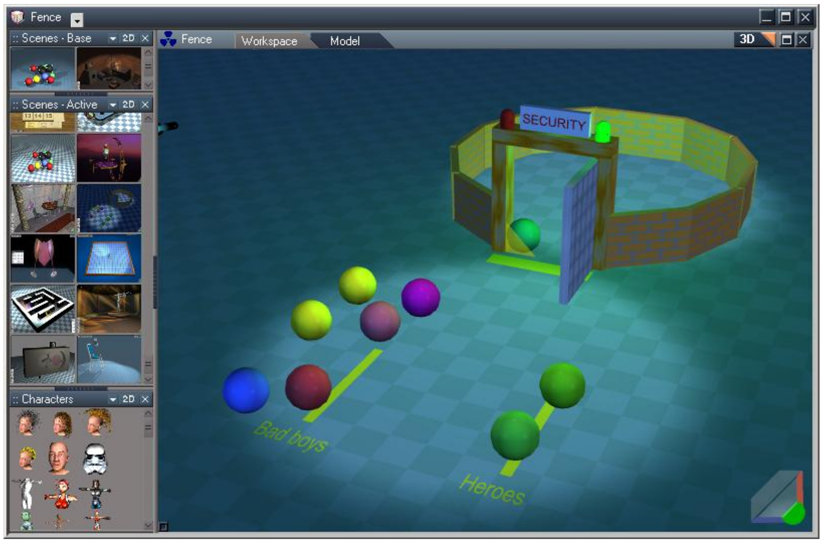
Load Secure Fence scene to start the activity.



Angle	1.047
AttConstant	1.000
AttLinear, 1	0.000
AttQuadratic, 1	0.000
ThresholdAtten	0.050
Color	
EnableShadow	

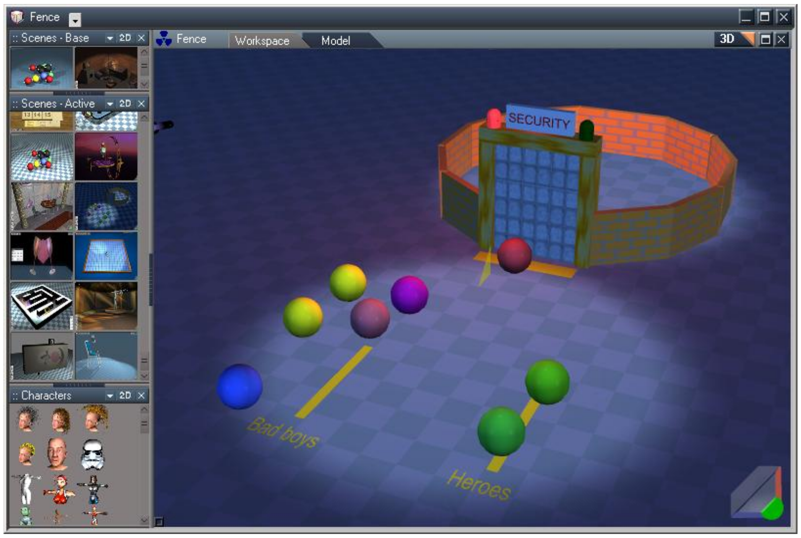
Loaded Secure Fence scene in the LE after switching to 2D.

Step 2: Select a Green Sphere and move it towards the doorway. The yellow line at the bottom of the doorway is the **Optical Detector**, which is tracking the sphere as you move it closer. When you get within "range" of the detector, the sphere's color is detected. If the color is green, the door is opened and the green light comes on.



Door opens for green sphere …green light is on

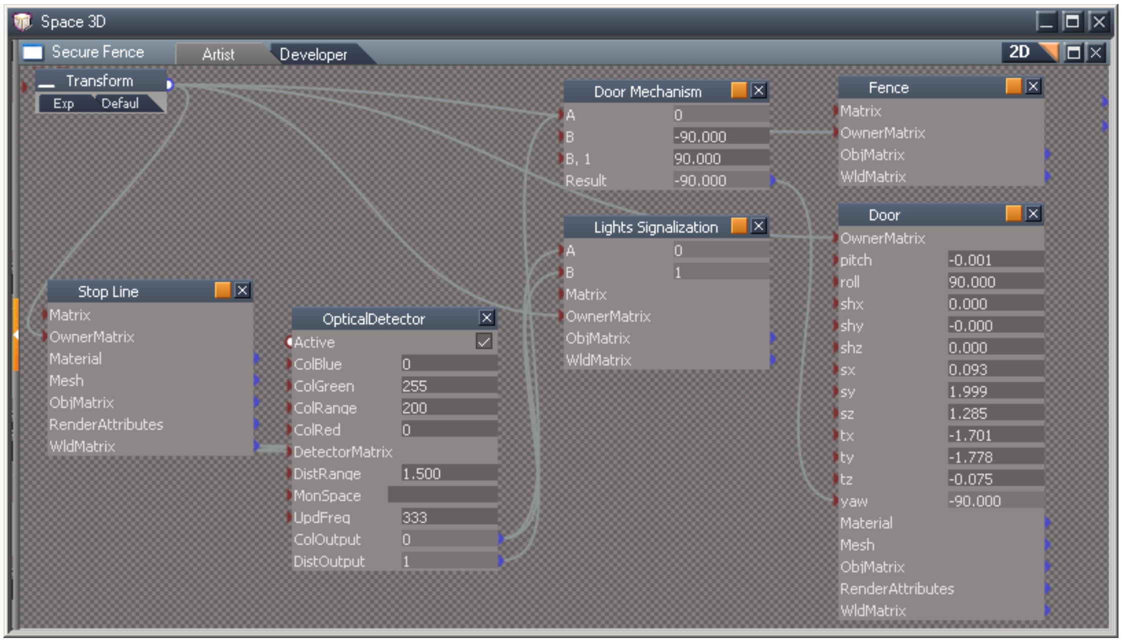
If the color is other than green, the door will not open and red lights indicate an unwelcome intruder.



….. but not for red one … red light is on

How does it work?

Let us take a look at what is involved in this more complex activity. In the Link Editor enter the Secure Fence object by left-clicking its orange entry square.



```
Door Mechanism
A        0
B        -90.000
B, 1     90.000
Result   -90.000

Fence
Matrix
OwnerMatrix
ObjMatrix
WldMatrix

Lights Signalization
A        0
B        1
Matrix
OwnerMatrix
ObjMatrix
WldMatrix

Door
OwnerMatrix
pitch    -0.001
roll     90.000
shx      0.000
shy      -0.000
shz      0.000
sx       0.093
sy       1.999
sz       1.285
tx       -1.701
ty       -1.778
tz       -0.075
yaw      -90.000
Material
Mesh
ObjMatrix
RenderAttributes
WldMatrix

Stop Line
Matrix
OwnerMatrix
Material
Mesh
ObjMatrix
RenderAttributes
WldMatrix

OpticalDetector
Active          ✓
ColBlue         0
ColGreen        255
ColRange        200
ColRed          0
DetectorMatrix
DistRange       1.500
MonSpace
UpdFreq         333
ColOutput       0
DistOutput      1
```

A look inside the Secure Fence object.

The yellow line in front of the door in the 3D window represents the **Stop Line** object. The line is connected to the **Optical Detector** object through the **Detector Matrix** attribute to determine its location in 3D space.

The **DistOutput** output attribute is automatically set to 1 if an object is nearer than or equal to the distance specified in the **DistRange** attribute. The **ColOutput** connector is set to 1 if the near object matches the color specified by the **ColRed, ColGreen** and **ColBlue** attributes. With the **ColRange** attribute you can specify the color deviation.

Finally, with the **Active** attribute you can activate and deactivate the **Optical Detector** from the outside level of the object as its connecter has been exported.

You can see in figure 3 that **ColOutput** is connected to the **Door Mechanism** so that it knows to open and close the door when the “right object” is detected. Similarly, both the output connectors are connected to the **Light Signalization** object. The lights indicate whether the object is allowed in or not.

11.4.2 PacMan

This is a simple 3D clone of the famous PacMan game, written in scripts. As always, PacMan tries to collect all of the gems while avoiding getting caught by the Ghost. You will get either a “congratulations” message at the end or “game over” with the score you achieved. You can start a new game by clicking “Press to start” again.
To start the game, open the load the Pac Man scene from the Scenes Active library:

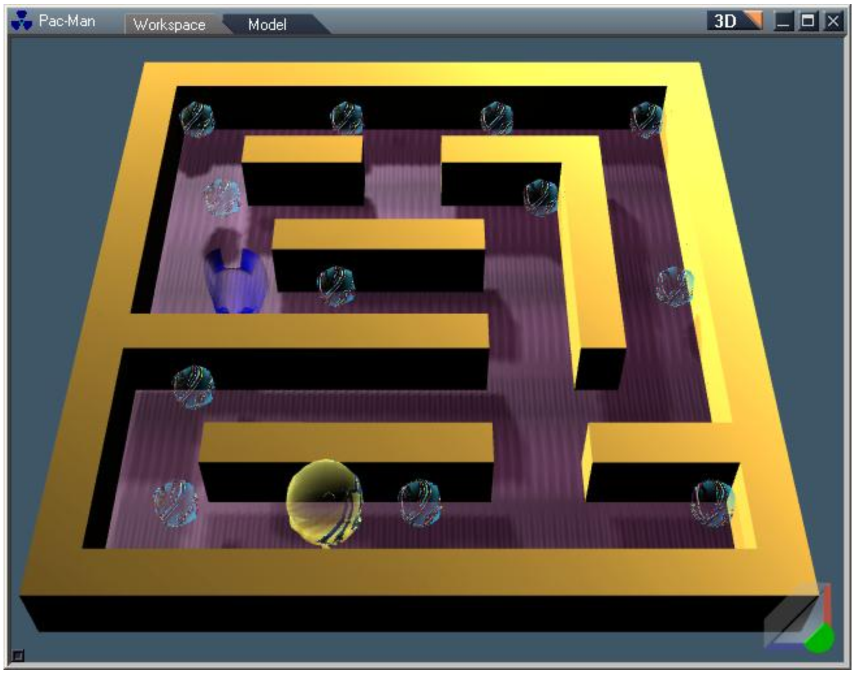
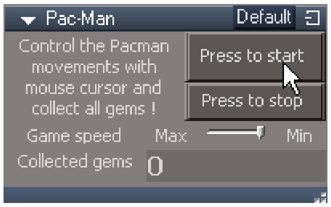


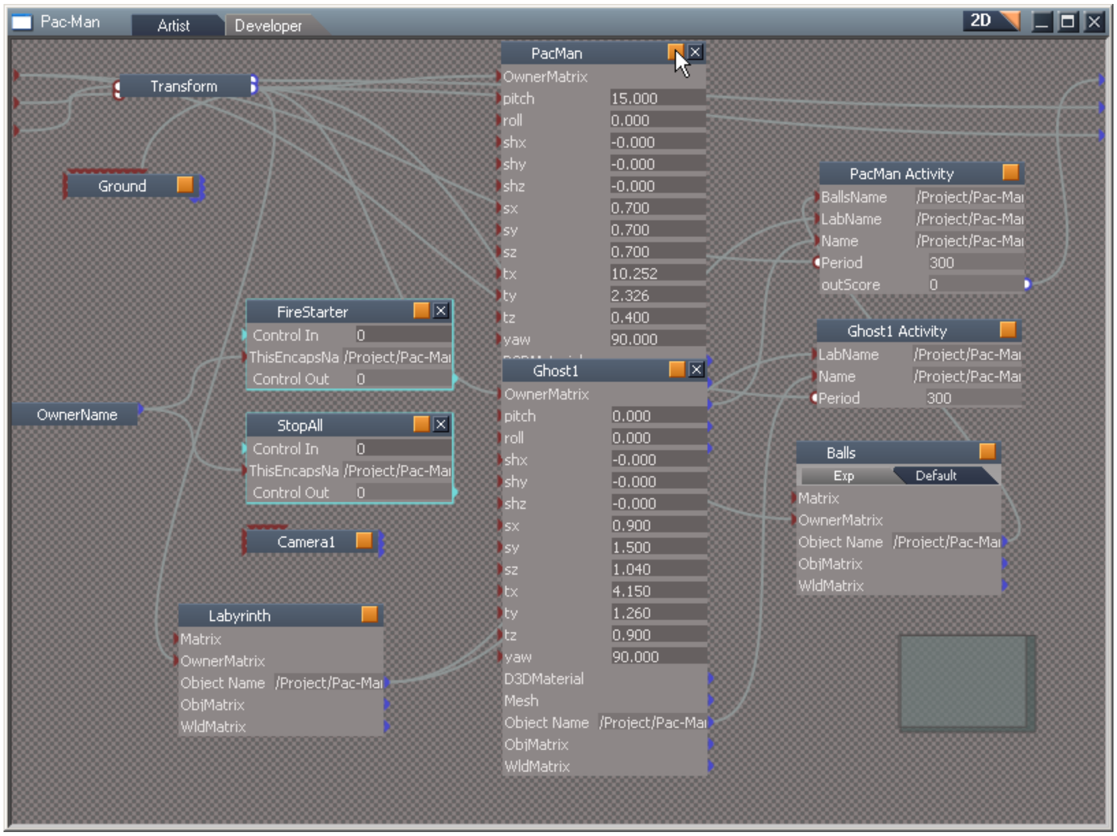
Figure 1: Pac Man activity.

Game controls:

- Click the “Press to start” button to start a new game.
- Click the “Press to stop” button to stop a current game at any time.
- the PacMan’s movements follow the position of the mouse cursor.



Game controls.



PacMan

OwnerMatrix	
pitch	15.000
roll	0.000
shx	-0.000
shy	-0.000
shz	-0.000
sx	0.700
sy	0.700
sz	0.700
tx	10.252
ty	2.326
tz	0.400
yaw	90.000

Ghost1

OwnerMatrix	
pitch	0.000
roll	0.000
shx	-0.000
shy	-0.000
shz	-0.000
sx	0.900
sy	1.500
sz	1.040
tx	4.150
ty	1.260
tz	0.900
yaw	90.000
D3DMaterial	
Mesh	
Object Name	/Project/Pac-Mai
ObjMatrix	
WldMatrix	

FireStarter

Control In	0
ThisEncapsNa	/Project/Pac-Mai
Control Out	0

StopAll

Control In	0
ThisEncapsNa	/Project/Pac-Mai
Control Out	0

Labyrinth

Matrix	
OwnerMatrix	
Object Name	/Project/Pac-Mai
ObjMatrix	
WldMatrix	

PacMan Activity

BallsName	/Project/Pac-Mai
LabName	/Project/Pac-Mai
Name	/Project/Pac-Mai
Period	300
outScore	0

Ghost1 Activity

LabName	/Project/Pac-Mai
Name	/Project/Pac-Mai
Period	300

Balls

Matrix	
OwnerMatrix	
Object Name	/Project/Pac-Mai
ObjMatrix	
WldMatrix	

Figure 2: PacMan object, entered in Link Editor.

Enter the PacMan object by left-clicking on its orange entry square in the Link Editor. You should see a screen similar the one above. This is the entire top-level structure of the PacMan game with links indicating relationships between important game objects. For example, both PacMan and Ghost have a relationship with the Labyrinth object.

To see more detail, you can enter any object in this top-level context that displays an orange square, which is like a door handle on the door to the next room.
Enter the Ghost1 Activity object:

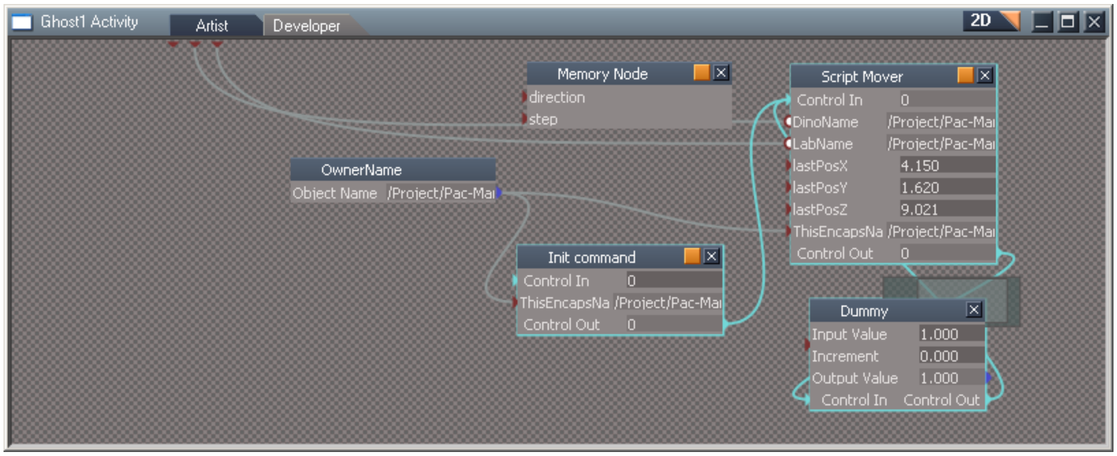
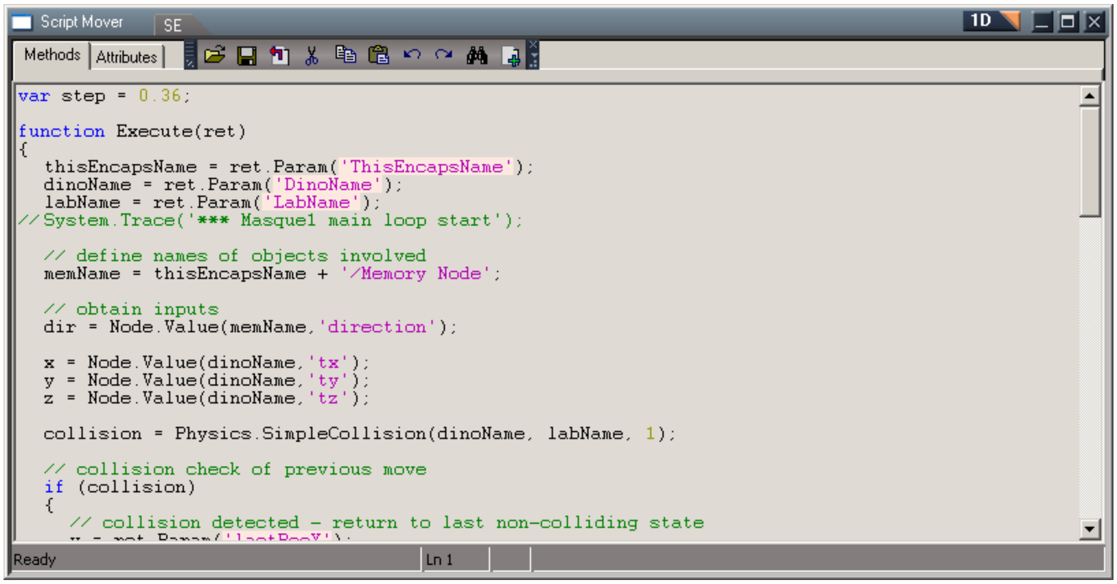


Figure 3: Ghost1 Activity object, entered in Link Editor.

Here you can see that Script Mover is an important part of the Ghost activity. You can also see some blue links that transfer control of activity between objects. Enter the Script Mover object next:



```
var step = 0.36;

function Execute(ret)
{
   thisEncapsName = ret.Param('ThisEncapsName');
   dinoName = ret.Param('DinoName');
   labName = ret.Param('LabName');
//System.Trace('*** Masquel main loop start');

   // define names of objects involved
   memName = thisEncapsName + '/Memory Node';

   // obtain inputs
   dir = Node.Value(memName,'direction');

   x = Node.Value(dinoName,'tx');
   y = Node.Value(dinoName,'ty');
   z = Node.Value(dinoName,'tz');

   collision = Physics.SimpleCollision(dinoName, labName, 1);

   // collision check of previous move
   if (collision)
   {
      // collision detected - return to last non-colliding state
      x = ret.Param('lastPosX');
```

Figure 4: Script Mover object entered in Link Editor, LE changes into Script Editor.

As you can see, Script Mover is written as a script, Javascript to be precise. Note that trueSpace automatically switched from 2D to 1D to present the appropriate editing controls for changing the structure of this object.

As it happens, several important objects in the PacMan game are written in script. They are mostly lower level objects. On the other hand, two top levels are created simply by linking these lower level objects together in the Link Editor to

create a complete game. The Link Editor allows you to navigate through all levels of an object easily and quickly, providing you with both a fast and a deep understanding of the entire game composition.
There is a also collaborative version of this game where one player gets to be a Pacman and the other one the Ghost.

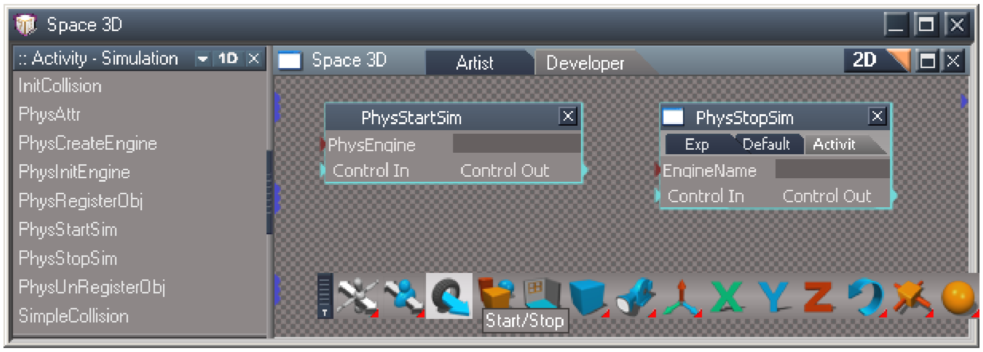
Finally, it is equally easy to take existing parts of PacMan and reuse them in a similar game simply by dragging and dropping.

11.5 Physics Activities

Activities can be combined with physical simulation using actions from the **Activity Simulation** library. There are five action objects contained in the library.

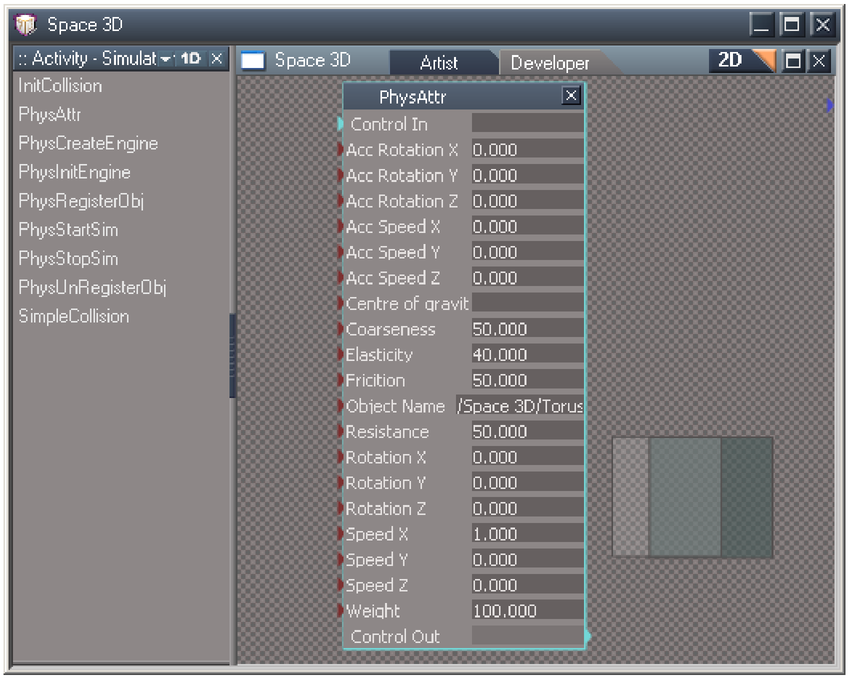
The actions *PhysStartSim* and *PhysStopSim* initialize and manage the global parameters of the simulation. *PhysStartSim* sets the basic time parameters of the physical engine, such as the total time of the simulation in seconds, and the time step, which determines the precision of the computation. (A higher value means higher precision.) *"Total time"* indicates the time interval in which the simulation will run. If you set it to 10, the simulation will run for 10 seconds. Setting the value to a number below zero (-1, for example) will cause the simulation to run non-stop. An Activity connector indicates whether the simulation is used with activities or not. The *PhysStartSim* engine is created in Space 3D and has the default name *PhysEngine* as is the name of the engine connector.

If you wish to stop a simulation before the simulation time has lapsed, simply push the simulation icon . You can manage the stopping of a simulation also by using the *PhysStopSim* object in the activity flow connection. In this case, however, you must specify the name of the engine (input connector *PhysEngine*) that you wish to interrupt.



Activities Simulation - start and stop objects

To simulate the behavior of objects using activities and physics, the objects must first have physical attributes assigned to them. You can do this by using the action object *PhysAttr*. There are several physical parameters that can be set for each object, including initial speed, initial rotation, weight, and elasticity.

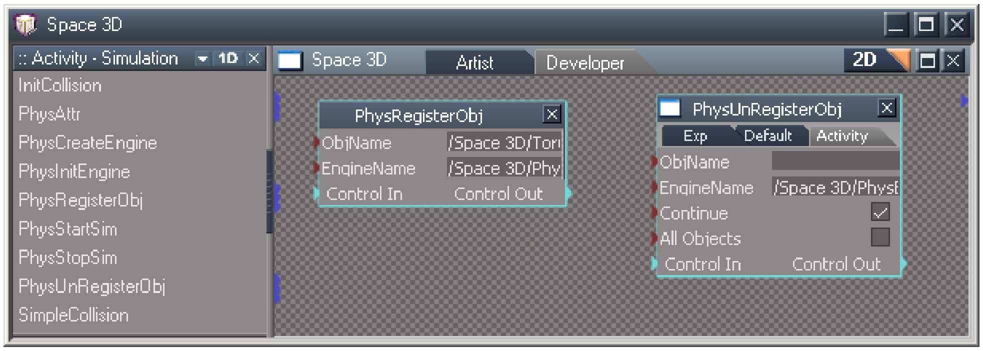


PhysAttr

Control In	
Acc Rotation X	0.000
Acc Rotation Y	0.000
Acc Rotation Z	0.000
Acc Speed X	0.000
Acc Speed Y	0.000
Acc Speed Z	0.000
Centre of gravit	
Coarseness	50.000
Elasticity	40.000
Fricition	50.000
Object Name	/Space 3D/Torus
Resistance	50.000
Rotation X	0.000
Rotation Y	0.000
Rotation Z	0.000
Speed X	1.000
Speed Y	0.000
Speed Z	0.000
Weight	100.000
Control Out	

Activities Simulation - PhysAttr object

The action object *PhysRegisterObj* starts the simulation of the object (connector *ObjectName*) using the physical engine (connector *EngineName*). On the other hand, the *PhysUnRegisterObj* object stops the simulation for a given object (connector *ObjectName*) by the specified engine using the connector *EngineName*. If the connector *All objects* is marked, all objects are excluded from the simulation.



PhysRegisterObj

ObjName	/Space 3D/Tor
EngineName	/Space 3D/Phy
Control In	Control Out

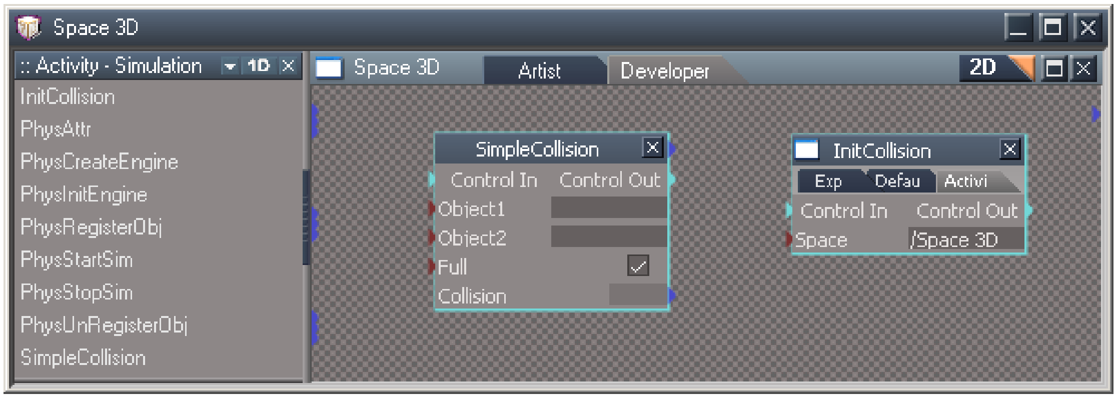
PhysUnRegisterObj

Exp Default Activity

ObjName	
EngineName	/Space 3D/Phys
Continue	✓
All Objects	☐
Control In	Control Out

Activities Simulation – PhysRegisterObj and PhysUnRegisterObj objects

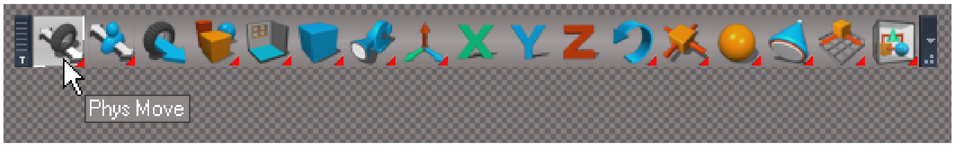
Other action objects, including *InitCollision* and *SimpleCollision*, are not directly linked to physical simulation and activities, but may be used in activities for collision detection between objects. The action *InitCollision* activates collision detection for a given space (identified in the attribute *“Space”)*. The *SimpleCollision* action object performs a check for collision between two objects with the names Object1 and Object2. If the objects collide, then the Collision attribute is set to 1; if they do not collide, the Collision attribute is set to 0. The attribute *Full* sets the precision level of collision checking. If *Full* is checked, a very precise collision analysis (on a mesh level) between objects is performed. If this attribute is not checked then only the collision between the bounding boxes of the objects is evaluated. If you wish to obtain collision information between an object and the ground, leave Object2 un-named or type in “ground” for Object2.



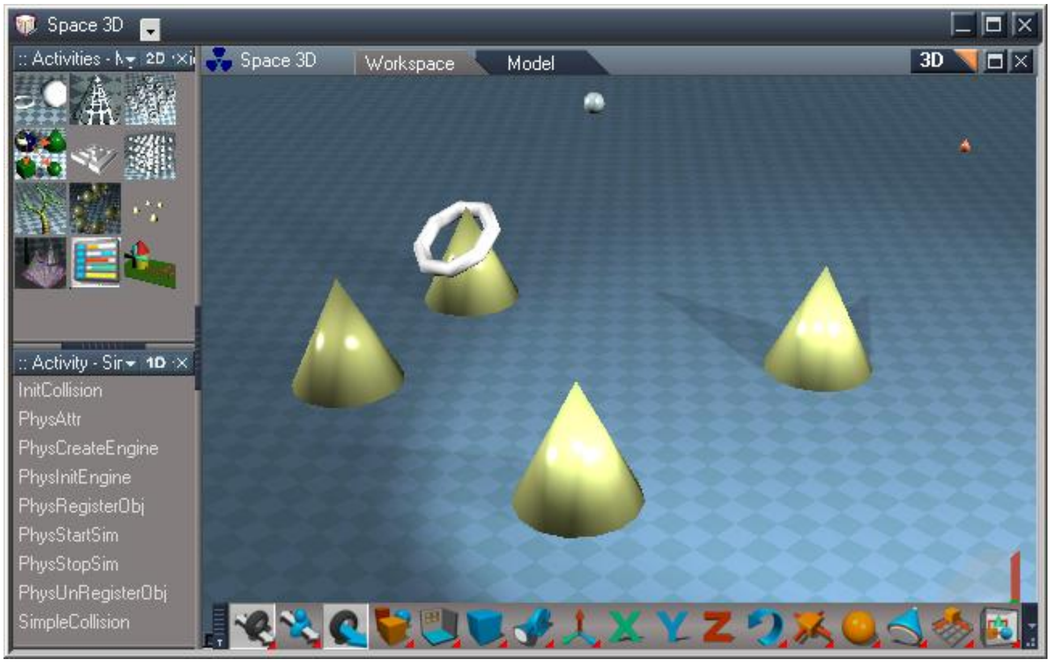
Action objects for collision checking

Physics Move Tool

You can use the **Physics Move Tool** during a simulation to set the position and speed of objects with physical attributes.
To interact with active objects during a simulation, you must select the **Physics Move Tool** from the Workspace toolbar. With this tool you can click and drag any physical object in the 3D scene. If an object has physical attributes and a simulation is running, a mouse drag sets the speed of the object, and the object will move according to your mouse movements. If you drag an object quickly, the speed of object will be fast, and so on.



Physics Move Tool in the Workspace toolbar



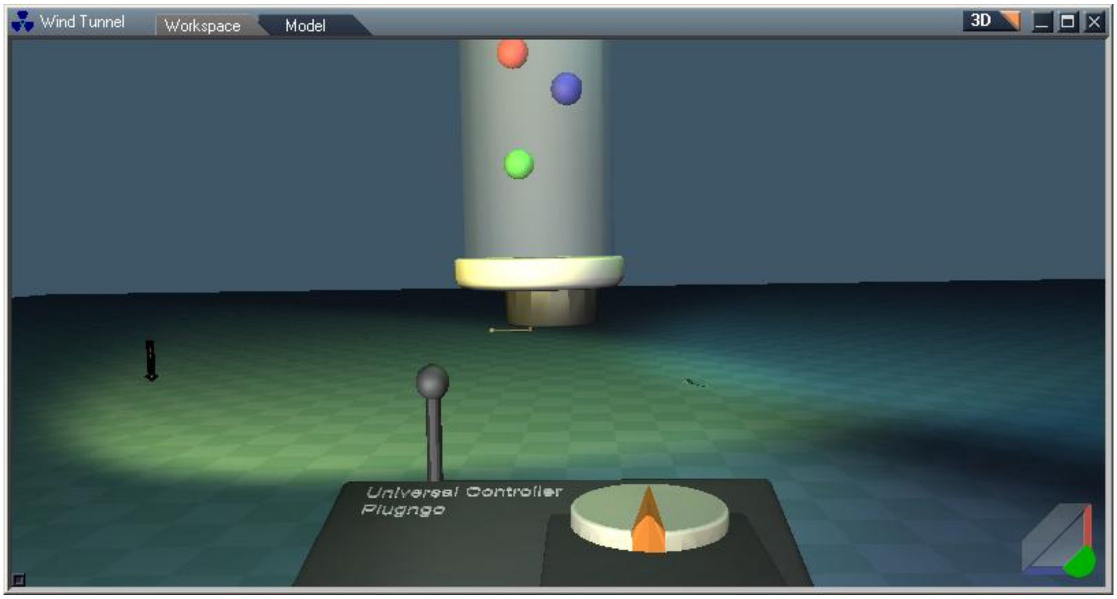
Using Physics Move Tool in the shot activity to place a torus on the cone.

Hint: Run the Shot activity example. "Shoot" a torus into the scene. Select the ***Physics Move Tool*** *from the navigation toolbar, click on the torus, and then try to place the torus on one of the cones by dragging the mouse.*

11.5.1 Tutorial: Wind Tunnel

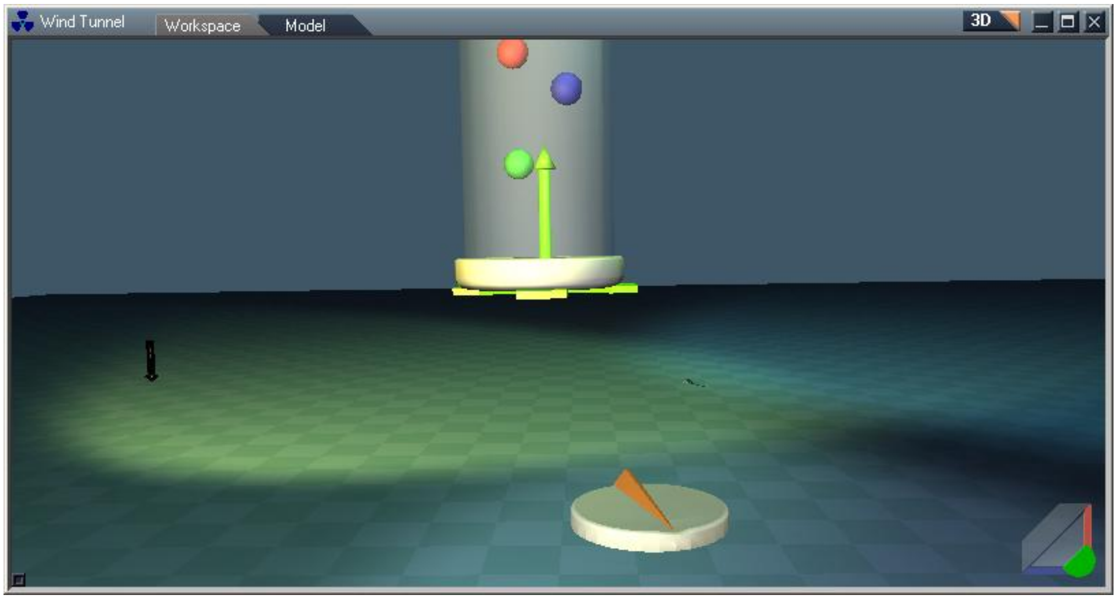
This tutorial will give you a look at how to create custom controls for any aspect of your scenes, and on how you can make your scenes truly interactive. You will build a small wind tunnel, which features three balls contained inside it. The strength of the wind is adjustable via a dial, which is a simple part of the scene. The wind strength can be set to range from having no effect, all the way up to blowing quite strongly! It will also be possible to reverse the wind direction.

All of this you will achieve without a single line of code or scripting, with just some simple use of the Link Editor to bring the scene to life.



The final scene

There are various elements that make up this scene. Some are important, while others are purely decorative. It is outside the scope of this tutorial to take you through the modeling steps to produce this scene, so we will just take a look at the individual elements and identify their purposes.
The completed scene is in the scene Libraries so you can load it and study how it was made before going on to make one of your own.

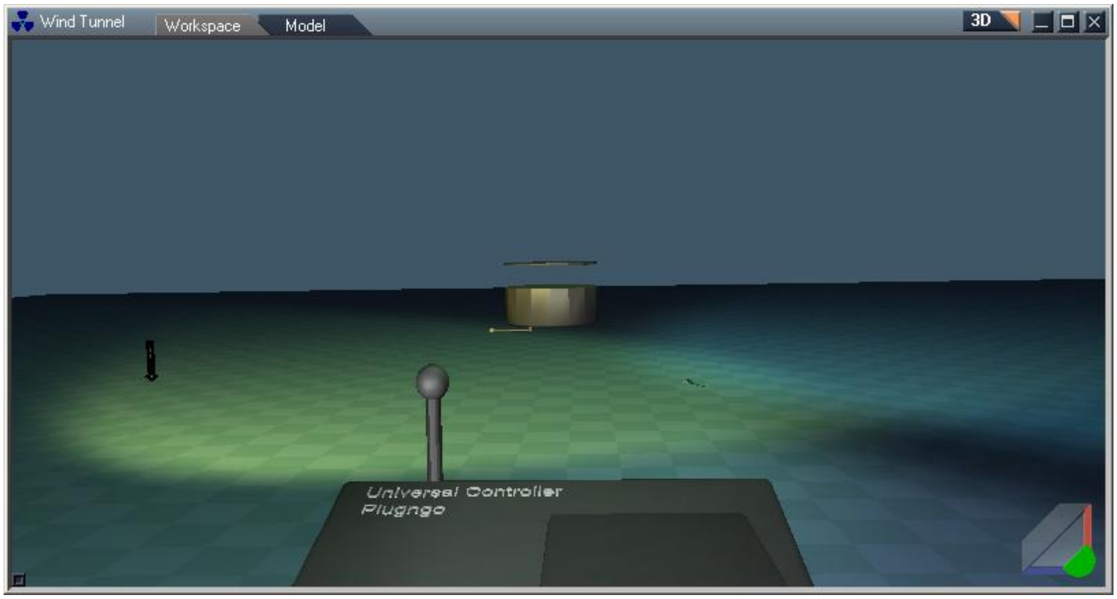


The important elements that play an active role in this scene

Here you can see the elements that play an active role in the scene. First there is the Local Wind object, indicated by a green arrow and fan. Next are the three spheres which will be influenced by gravity and the Local Wind once you start physics.

The main element in this scene which makes it unique from most scenes in a 3D modeling package is the dial. This is a regular 3D object, just like any other in your scene, but you are going to use the Link Editor to connect the rotation property of the dial to the strength of the Local Wind. This is going to turn the dial into a control for an aspect of the scene, while the control is still part of the environment itself, as if you were running a game engine.

The cylinder is a vital element to this scene, as it will keep the balls in place; without it, they would simply blow away under the influence of the Local Wind, probably before we could experiment much with our dial to see what happened! It is made from three objects: a hollow cylinder, and two end caps. You can make any sort of container you like, of course.



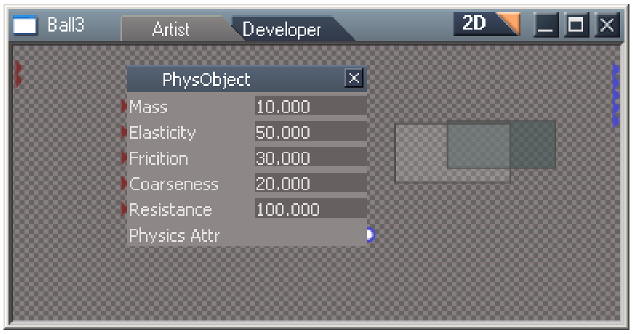
Purely decorative objects that give the scene some meaning and bring it to life

These last objects don’t play any role in the physics at all. They have been added in there to give the scene more substance and anchor it in reality. You could go further of course, and construct a laboratory room with a desk for the objects to sit on, and so forth. Here, the objects are a little motor on the end of the tube, and a remote control which will house the dial. This sets up the scene as if you were really interacting with physical objects, making it less abstract.

Making It Happen

We’re going to start from the point where you have all your objects modeled and positioned, including your Local Wind.

The first step is so assign Physical Attributes to the balls. Note that you will most likely need to increase the Resistance and decrease the Mass and Friction of the balls to produce a more interesting effect. Some sample parameters are shown below.



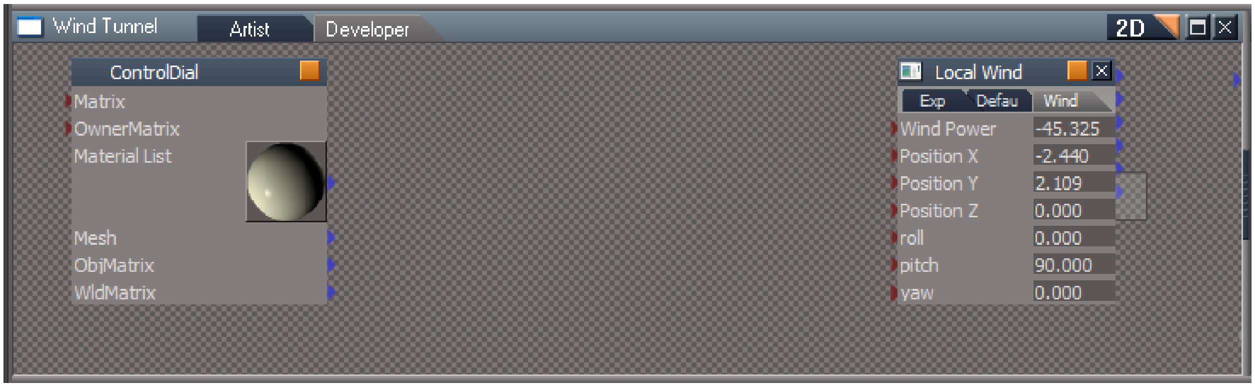
Mass	10.000
Elasticity	50.000
Fricition	30.000
Coarseness	20.000
Resistance	100.000
Physics Attr	

Sample Physical Attributes

Note that your other objects such as the cylinder do not need Physical Attributes assigned. The balls will still collide with them, which is what we want. Assigning Physical Attributes is only necessary for those objects that are going to move under Physics.

Press the Physics button, and your balls will be blown by the wind, fall down under the influence of gravity, and bounce off the inner walls of the cylinder.

Let’s turn our scene into something that is not normally possible in a 3D modeling application: an interactive scene like you might find in a game engine. To do this, we will need to hook up the Dial to the Local Wind.

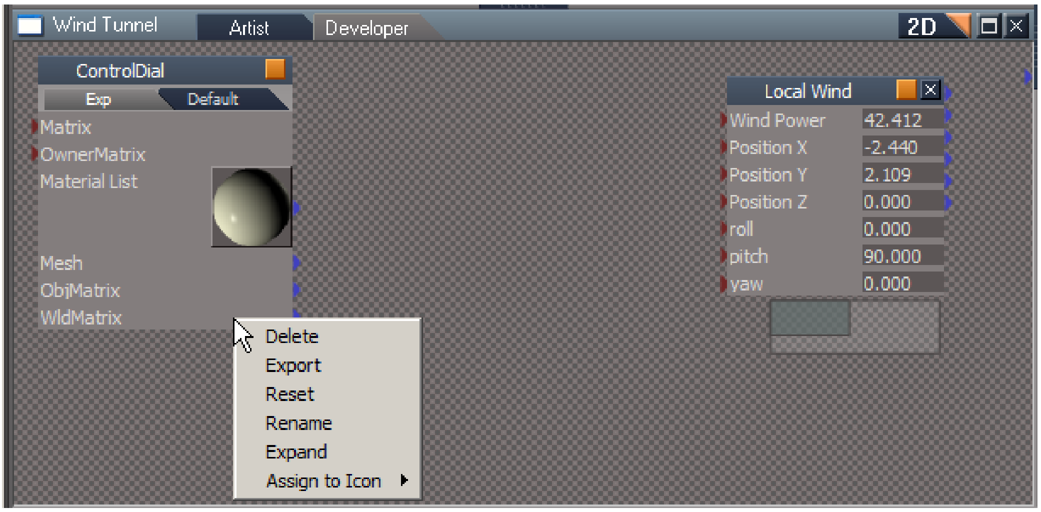


Wind Power	-45.325
Position X	-2.440
Position Y	2.109
Position Z	0.000
roll	0.000
pitch	90.000
yaw	0.000

The Dial and the Local Wind objects, as yet unconnected

We want the rotation of the Dial to affect the strength of the wind. You can see the Wind Power parameter in the Local Wind object since it is a parameter that is displayed in the Default aspect of that object. The rotation of the Dial, though, is hidden, so you will need to make it visible.

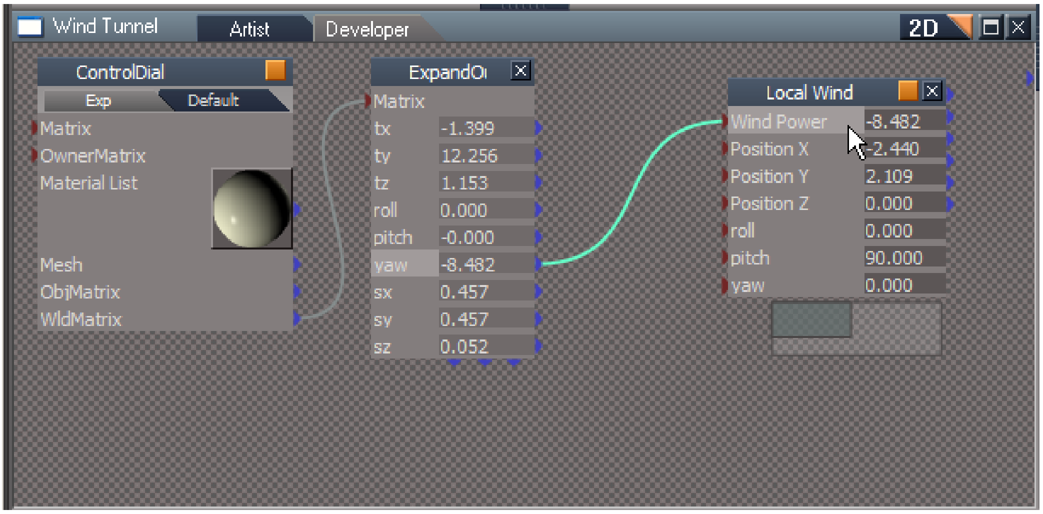
The quick and easy way is to right-click on the WldMatrix parameter and select Expand. This instantly opens up a new object in the Link Editor which shows us all the parameters of the Dial object, such as its size, position, and rotation, as seen below.



Local Wind	
Wind Power	42.412
Position X	-2.440
Position Y	2.109
Position Z	0.000
roll	0.000
pitch	90.000
yaw	0.000

Reveal the parameters of the Dial object

In this instance, we are interested in the Yaw parameter. (If you are unsure of which one to use, you can simply try rotating the Dial object in the Workspace using the widget and watch which parameter changes.)

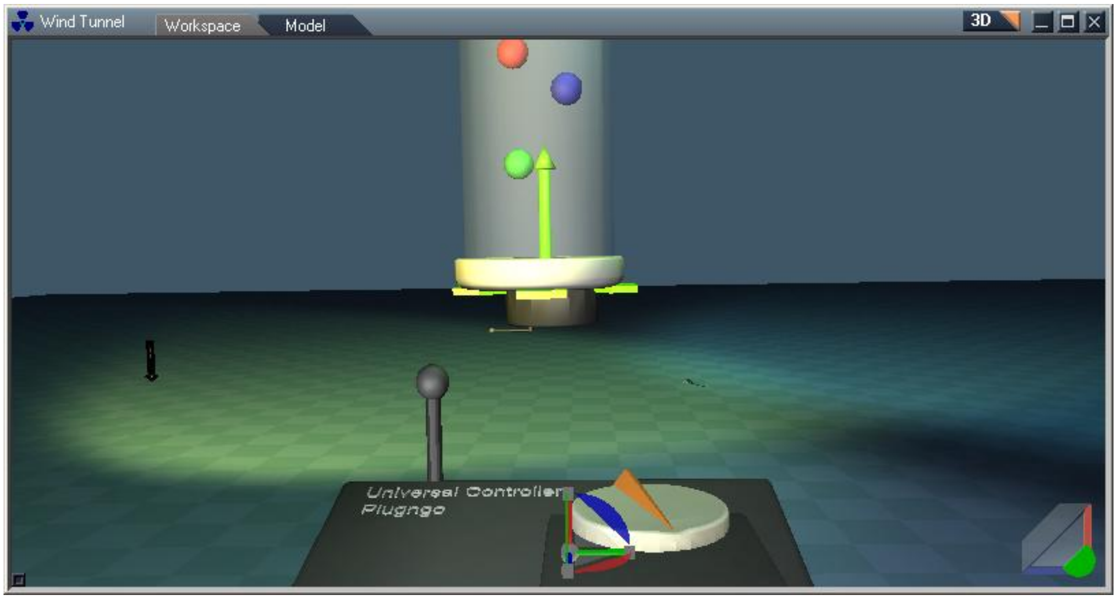


ExpandOı	
tx	-1.399
ty	12.256
tz	1.153
roll	0.000
pitch	-0.000
yaw	-8.482
sx	0.457
sy	0.457
sz	0.052

Local Wind	
Wind Power	-8.482
Position X	-2.440
Position Y	2.109
Position Z	0.000
roll	0.000
pitch	90.000
yaw	0.000

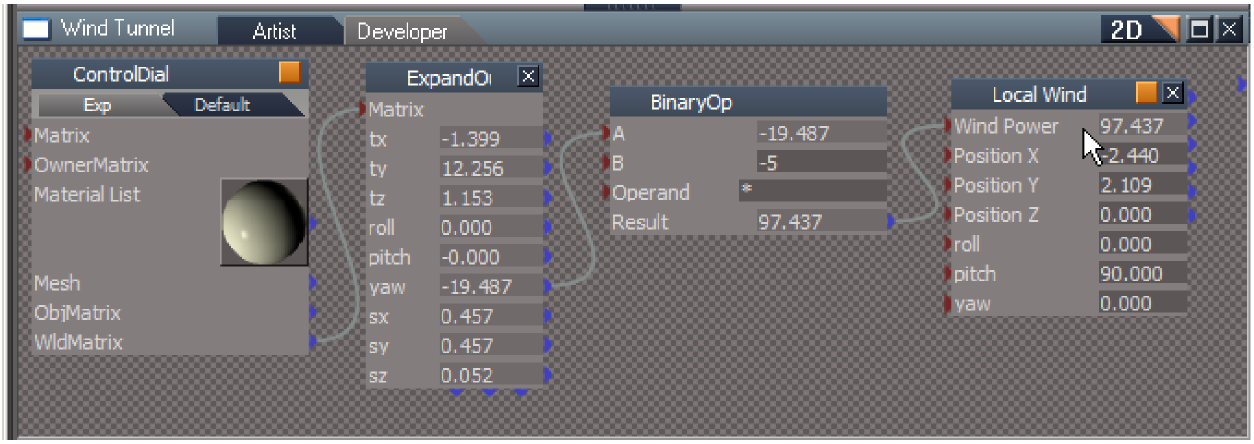
Connect the Yaw to the Wind Power, and your Dial becomes an interactive control

Now we will connect our Dial to the Local Wind. Click on the blue arrow beside the Yaw, and drag to the Wind Power attribute in PhysWind to create the link. With this single step, you have taken a regular trueSpace scene and turned it into an interactive trueSpace scene. When you turn the Dial, you will change the strength of the Local Wind, and change what is happening in the scene. Because you are using the built-in Physics and the built-in geometry, you did not need to write a single line of code or script to get this to happen. Let's take a look at how to interact with the scene in detail.



Rotate the red arc on the object widget for the Dial to control the wind strength

Using the Dial is simple. We want to rotate it, and the easiest and most controllable way to do that is to use the 3D object widget.
As you rotate the dial, you will change the strength of the Local Wind. You can see this in the Wind Power box in the Link Editor,. The image above illustrates the effect. Note that Physics is not running in this screenshot, which is why the balls remain at their starting positions.



ControlDial (Exp / Default)
- Matrix
- OwnerMatrix
- Material List
- Mesh
- ObjMatrix
- WldMatrix

ExpandOı

Matrix	
tx	-1.399
ty	12.256
tz	1.153
roll	0.000
pitch	-0.000
yaw	-19.487
sx	0.457
sy	0.457
sz	0.052

BinaryOp

A	-19.487
B	-5
Operand	*
Result	97.437

Local Wind

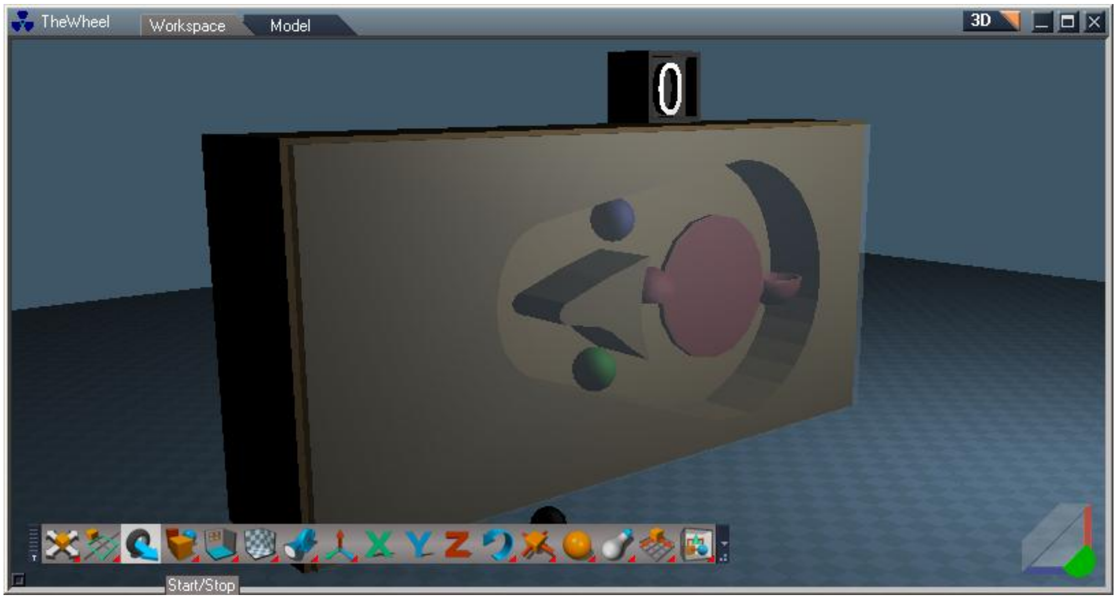
Wind Power	97.437
Position X	-2.440
Position Y	2.109
Position Z	0.000
roll	0.000
pitch	90.000
yaw	0.000

Rotating the Dial adjusts the Power of the Local Wind in the LE

The BinaryOp which is found in the Math - Operations library is being used so that the result from rotating the dial follows the common way for increasing a value when adjusting a dial , without it the results would become negative values as the dial was turned clockwise and the power would decrease as the dial was turned.
That brings us to the end of this tutorial, which we created without a single line of code or script. Of course, it would be easy to use the same principles for many other scenes. You could connect the rotation of the dial to the strength of gravity, either globally or for a Local Environment, or to the intensity of a light, or to the position of a robot arm. You could also change the object from a dial to a slider in the 3D space, or to a lever or switch, and in this way you could construct interesting and complex scenes with a high level of interactivity.

11.5.2 Tutorial: The Wheel

In this tutorial we will work with a scene that has a machine-like element to it, with a wheel that rotates constantly as if driven by a motor. This wheel moves balls inside the machine, which in turn trigger a scoring system when they pass by a certain point. All of this is done using the physical simulation abilities of trueSpace, and by assembling existing objects in the Link Editor without writing any code or scripting.
The completed scene can be found in the scenes libraries so you can load it and study as you go along.



The final scene – click Run Physical Simulation to see what happens

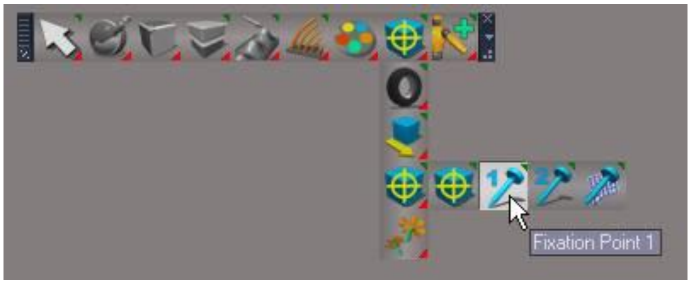
This “desktop toy” object serves no real purpose, but it is fun to watch! The wheel and balls are enclosed in a case, which has a transparent top cover. Whenever a sphere passes through the channel, it triggers a light and adds one to the counter. All the elements and ideas in this scene can easily be used in many other scenes.

Setting The Scene

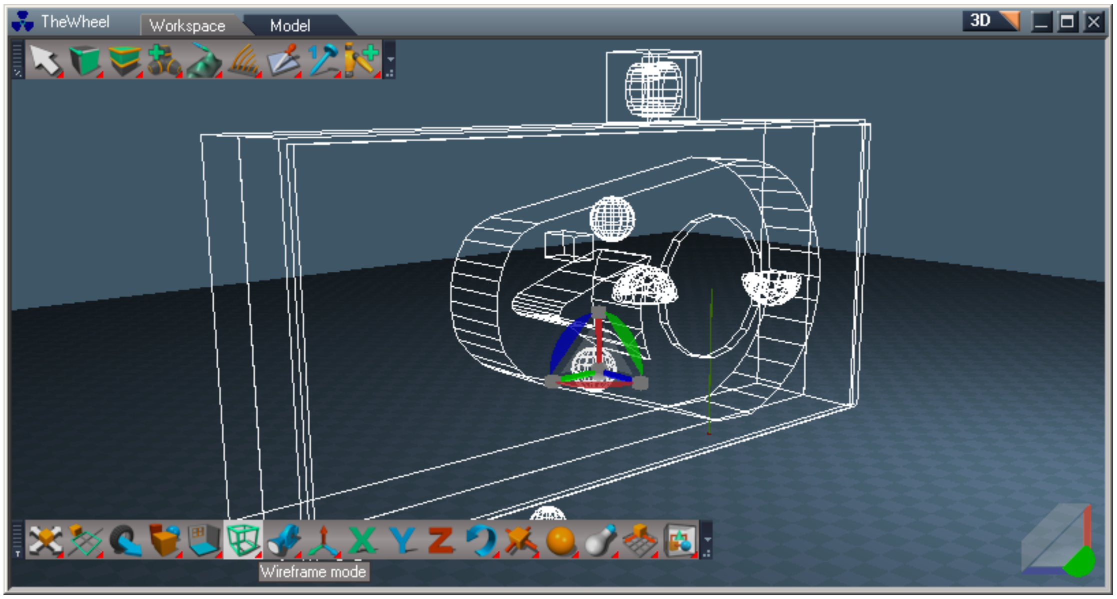
The scene is easily constructed. The case is a set of simple objects, with a Solid Transparency material used for the cover. The spheres are simple primitives with Physical Attributes assigned to them. Changing the Physical Attributes such as Mass, Elasticity, Friction, Resistance, and so on, will give different results for the machine, so have fun experimenting!

You can also experiment with changing the physical parameters, too, since changing the gravity, atmosphere, and accuracy all affect the animation that plays out.

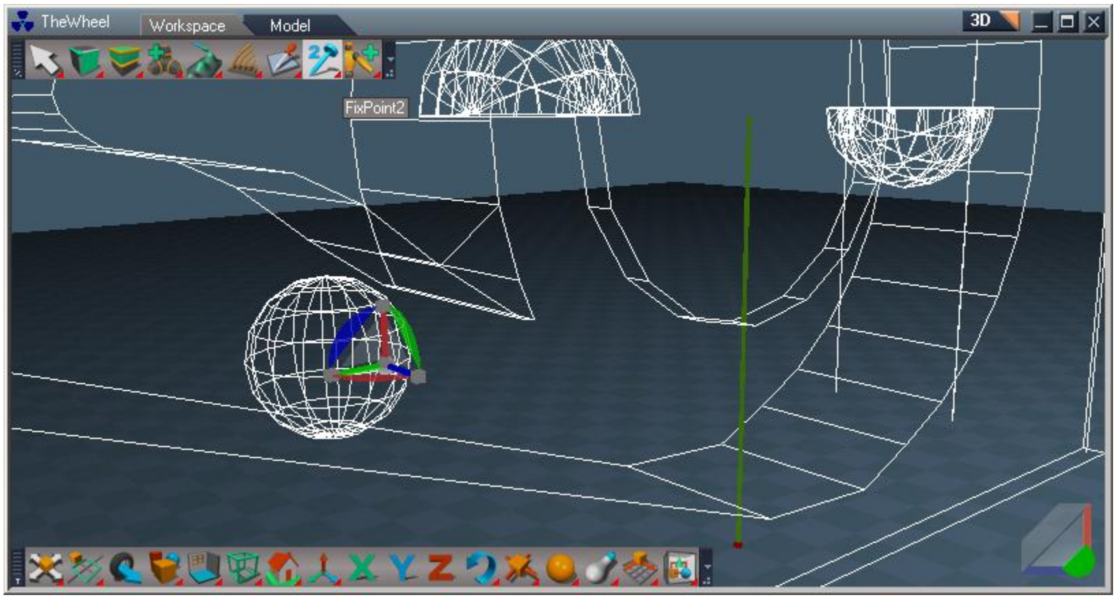
The wheel itself rotates under its own power using Physical Simulation alone, without any keyframing or scripting. The first thing that needs to be set up are some constraints, called Fixation Points. Click on the Add Phys Attr icon to open up the physics toolbar, and by selecting the wheel object and choosing FixPoint 1 and FixPoint 2, we can assign two “nails” to the wheel that will limit its movement. This will prevent it from falling down under gravity or rotating in any direction other than the one we want.



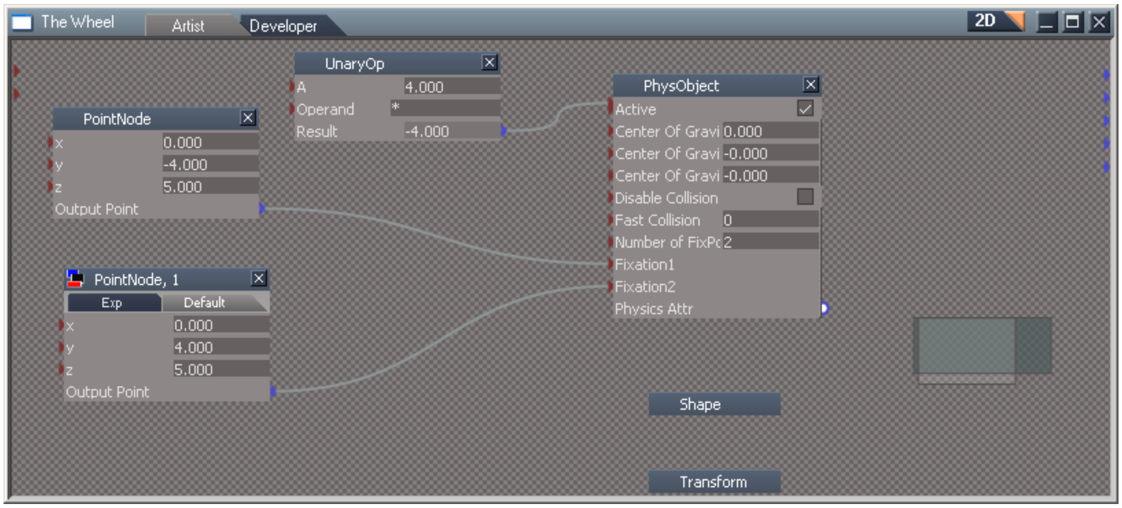
Location of fixation point tool



Workspace changed to wire mode to see the physics widgets more clearly



Zooming in to see the physics fixation point widgets



```
UnaryOp
A              4.000
Operand        *
Result         -4.000

PhysObject
Active                ✓
Center Of Gravi  0.000
Center Of Gravi  -0.000
Center Of Gravi  -0.000
Disable Collision
Fast Collision   0
Number of FixPc  2
Fixation1
Fixation2
Physics Attr

PointNode
x              0.000
y              -4.000
z              5.000
Output Point

PointNode, 1
Exp            Default
x              0.000
y              4.000
z              5.000
Output Point
```

Setting the nails precisely in the Link Editor

Above you can see the fixation points displayed in the Workspace. In this instance, the values were set in the Link Editor to position them accurately. Simply enter the wheel object to view and work with this information directly in the point node objects. What is important here is that the two nails lie in a straight line, one on either side of the wheel.

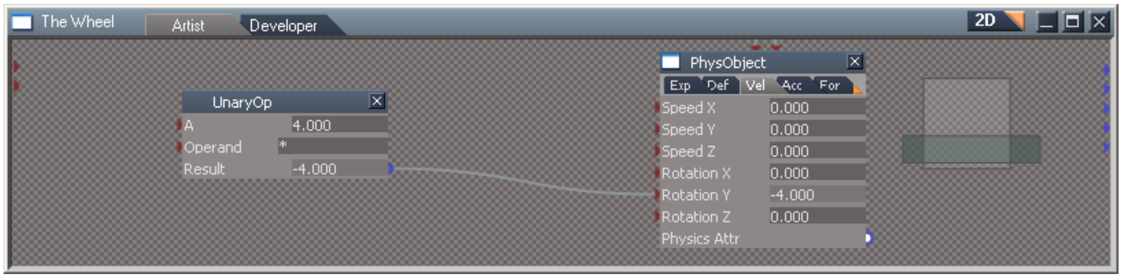
With the basics covered, the next step is to introduce the elements that will bring the scene to life.

Another Turn of the Wheel

You can make an object rotate by setting a value for one of the Rotation attributes (found in the physics settings inside the object). However, that value will change during simulation. For example, the wheel will slow down due to friction or air resistance.

Removing resistance or friction would alter how the wheel interacts with the balls, so we do not want to do that. Instead, we can "drive" the wheel by constantly feeding in a value for the rotation so that it gets reset despite physical elements acting on it.

By adding in a UnaryOp (unary operator, found in the Math Components, Operations library) as seen in the image below, we can constantly feed a value into the rotation parameter. This produces a realistic situation: the UnaryOp object will try to keep the wheel rotating at a constant speed, just as a motor might in the real world. Collision with the balls will cause the wheel to try to slow down, and if a ball gets trapped between the wheel and the case, the wheel will become "jammed" (unlike using keyframes, where the wheel would just move straight through the ball without any interaction at all).



UnaryOp
A 4.000
Operand *
Result -4.000

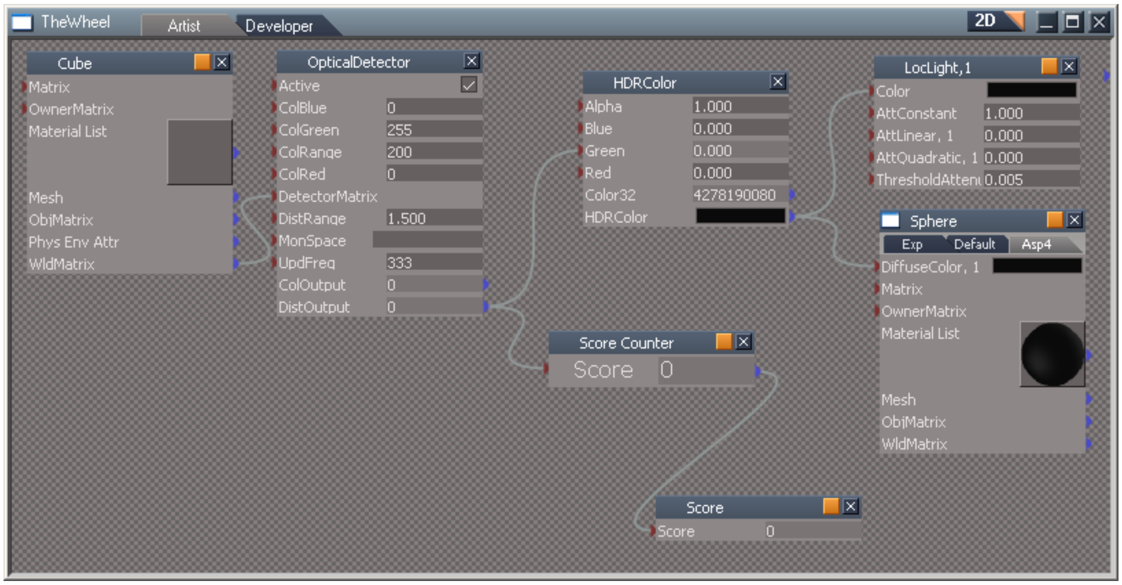
PhysObject
Exp Def Vel Acc For
Speed X 0.000
Speed Y 0.000
Speed Z 0.000
Rotation X 0.000
Rotation Y -4.000
Rotation Z 0.000
Physics Attr

Driving the wheel's rotation

Using a UnaryOp like this would have similar results in many physical situations, acting like an engine or motor which attempts to move an object in a constant manner, but not always succeeding depending on the physical interactions that take place.

The Watcher

To make things a little more interesting, there is a detector which will spot when an object moves past. This object is a copy of the Optical Detector from the Security Door scene (which you can find in the Building Blocks library). Re-use objects from other scenes or libraries to save yourself from reinventing the wheel.



Cube
Matrix
OwnerMatrix
Material List
Mesh
ObjMatrix
Phys Env Attr
WldMatrix

OpticalDetector
Active ✓
ColBlue 0
ColGreen 255
ColRange 200
ColRed 0
DetectorMatrix
DistRange 1.500
MonSpace
UpdFreq 333
ColOutput 0
DistOutput 0

HDRColor
Alpha 1.000
Blue 0.000
Green 0.000
Red 0.000
Color32 4278190080
HDRColor

LocLight,1
Color
AttConstant 1.000
AttLinear, 1 0.000
AttQuadratic, 1 0.000
ThresholdAtten 0.005

Sphere
Exp Default Asp4
DiffuseColor, 1
Matrix
OwnerMatrix
Material List
Mesh
ObjMatrix
WldMatrix

Score Counter
Score 0

Score
Score 0

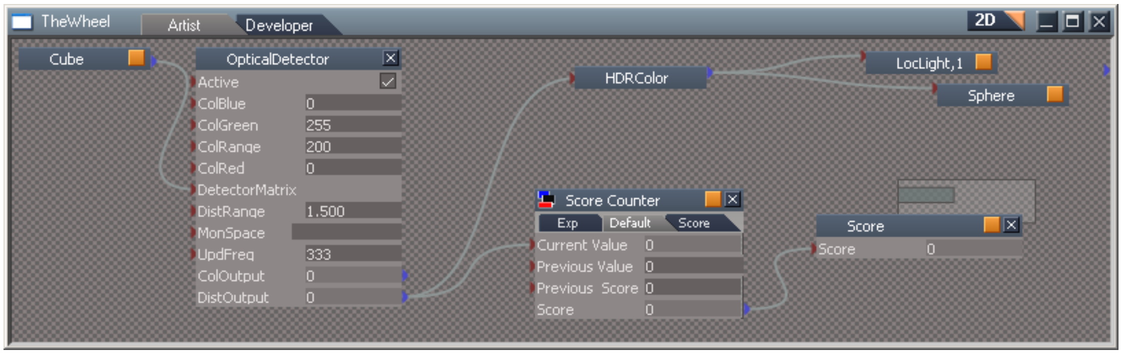
Detecting when a ball passes nearby

A cube is created painted with a totally transparent material to make it invisible in the scene, and then positioned to act as the area that is checked by the Optical Detector. The value of the DistRange attribute of the Detector is set to a small enough value to avoid detecting the case and only trigger when something comes closer. The color range attributes are set so that any ball will trigger the count. You can adjust these color and range parameters yourself to change the workings of the machine, so that it counts only red balls, or to narrow the range at which it detects an object, for instance.

Whenever something passes close enough to the cube, the DistOutput attribute on the Detector becomes set to 1; otherwise, it remains at 0 if there is no object nearby. This DistOutput value is passed on to the Green attribute of an HDRColor object (from the DXComponents, View library), and that color is then fed into a light and a sphere. This is an easy way of making the light change from black to green as the detector picks up on a passing object.

Keeping Count

The final step is to count how many times a ball comes down the extra channel. This can be done by hooking up a Score Counter object and connecting the result from that to the Score object, which will display the score. You can find both the Score Counter and the Score object in the Building Blocks library.



OpticalDetector
Active ✓
ColBlue 0
ColGreen 255
ColRange 200
ColRed 0
DetectorMatrix
DistRange 1.500
MonSpace
UpdFreq 333
ColOutput 0
DistOutput 0

Score Counter
Exp | Default | Score
Current Value 0
Previous Value 0
Previous Score 0
Score 0

Score
Score 0

Expanding the aspect of the Score Counter

The Score Counter works by incrementing each time its input changes to a number greater than the previous input. This means when the Detector changes from 0 to 1, the Score Counter adds 1 to its score. (When the Detector changes back from 1 to 0, the Score Counter ignores the change.) The connections are illustrated in the Expanded aspect above. Simply connect your value to the Current Value attribute, and the object takes care of the rest. You can reset it at any time by entering a new value into the Previous Score attribute. (Both Previous Score and Previous Value are the “memory” for the object to store values over time.)

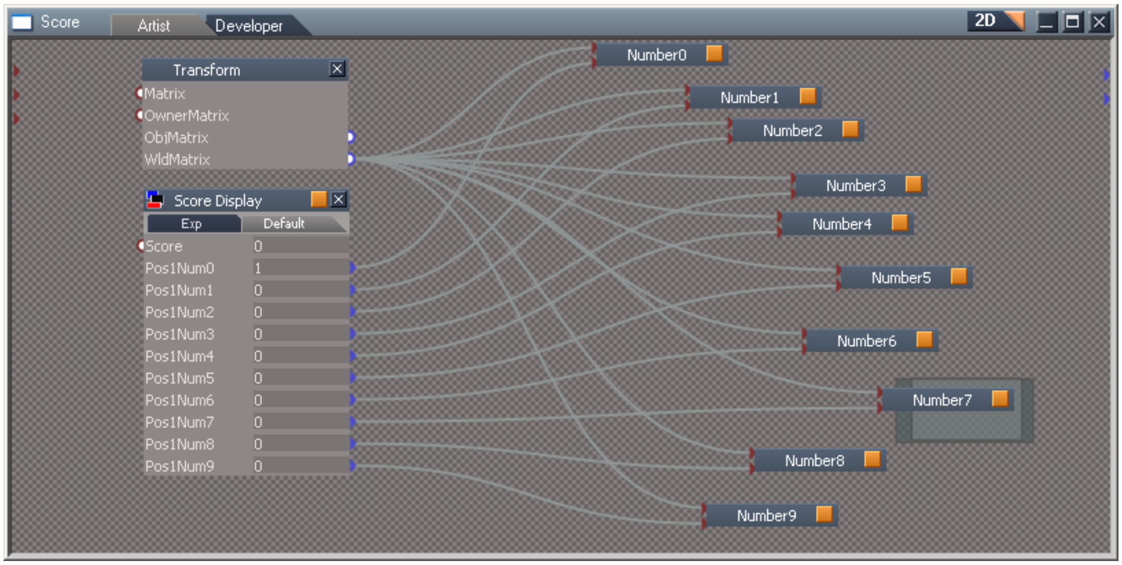
Once you have hooked up the input, you can switch to the Score aspect, which is simpler to use because it hides the various “memory” parameters and only displays the one thing of interest, the current score.

The Score Counter is a simple script that you can enter and edit. For example, you could change it to detect particular conditions or values being sent to it and how it decides to when to increase the score.

Displaying the Score

The final step for this scene is to display the score in the 3D scene itself. The Score object does this. All it requires is an input, and it displays the values from 0 to 9 in the 3D scene, updating in the Workspace automatically whenever a new value is sent to it.

Its operation is deceptively simple. It has 10 geometric objects for the digits 0 through to 9, and it changes the size of those objects in order to control which is visible. So if it receives the value 3, it ensures that the size of the geometric objects for 0, 1, 2, 4, 5, 6, 7, 8 and 9 are set to 0 (effectively making them disappear), while the size for the geometric object for 3 is set to 1.



Score	0
Pos1Num0	1
Pos1Num1	0
Pos1Num2	0
Pos1Num3	0
Pos1Num4	0
Pos1Num5	0
Pos1Num6	0
Pos1Num7	0
Pos1Num8	0
Pos1Num9	0

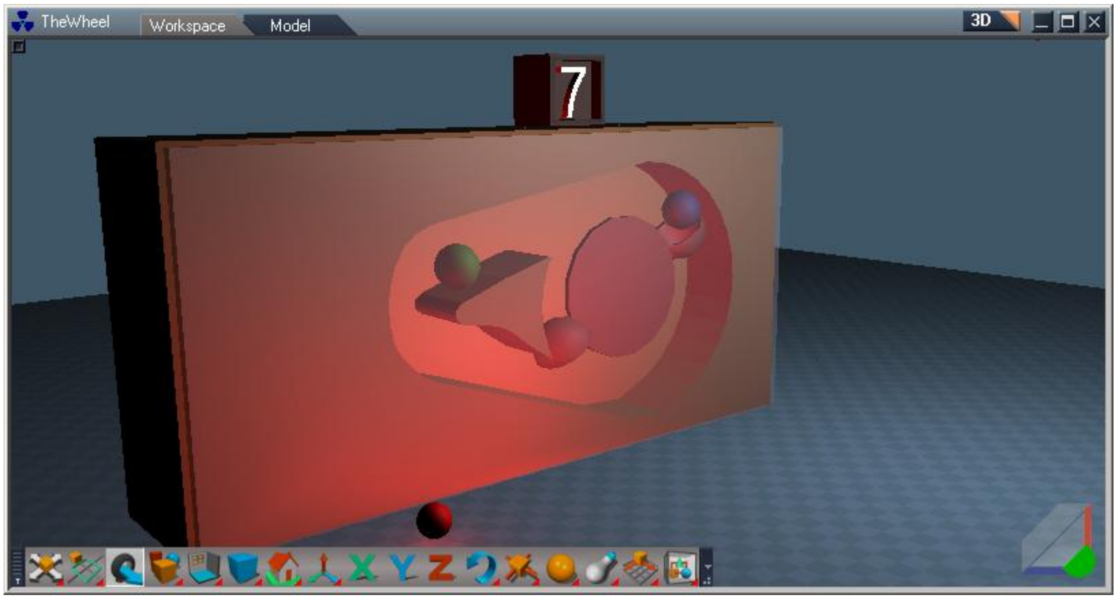
Looking inside the Score object

Looking inside the Score object will reveal quite a few links, but if you keep in mind the description from above on how it works, you will see it is much simpler than it looks.
For the curious, the Score Display object is a script that handles setting the values for the 10 output attributes, one for each digit. This value is then passed along to control the size of each geometric object. Most of the settings are used to enable the correct display of the geometry, and to ensure you can select the whole Score object and move it into position without having to move each digit individually. You retain the ability to adjust the size, rotation, scale etc of each digit independently, too, just like a regular hierarchical object.

Things To Try

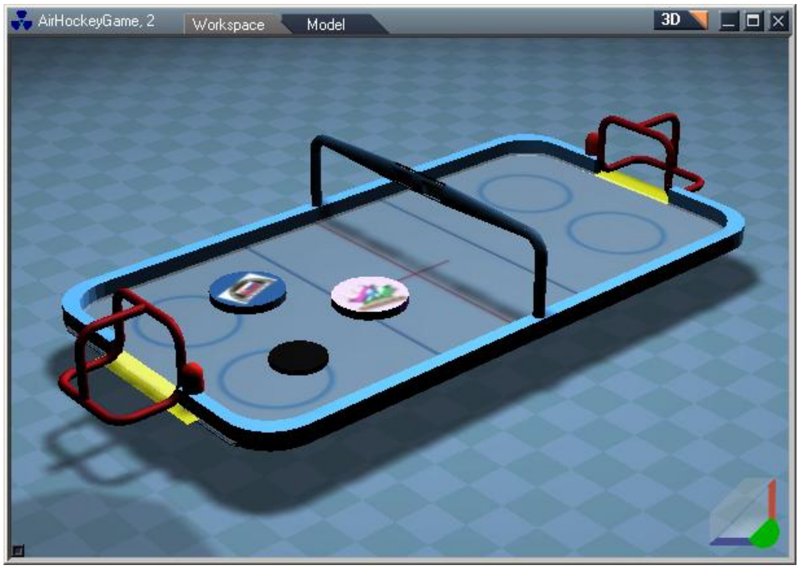
Now that we have explored how this scene was created, here are some things you might like to try. Remember, you can also copy and reuse the objects from this scene in creations of your own, too.

- Adjust the physical attributes of the balls, the physics settings for the whole scene, or the speed of the wheel.
- Build a new case and place the inner workings of the machine in there.
- Add more balls.
- Change the color of the light.
- Only count score for a certain color of ball.
- Advanced: Change the font of the score counter.



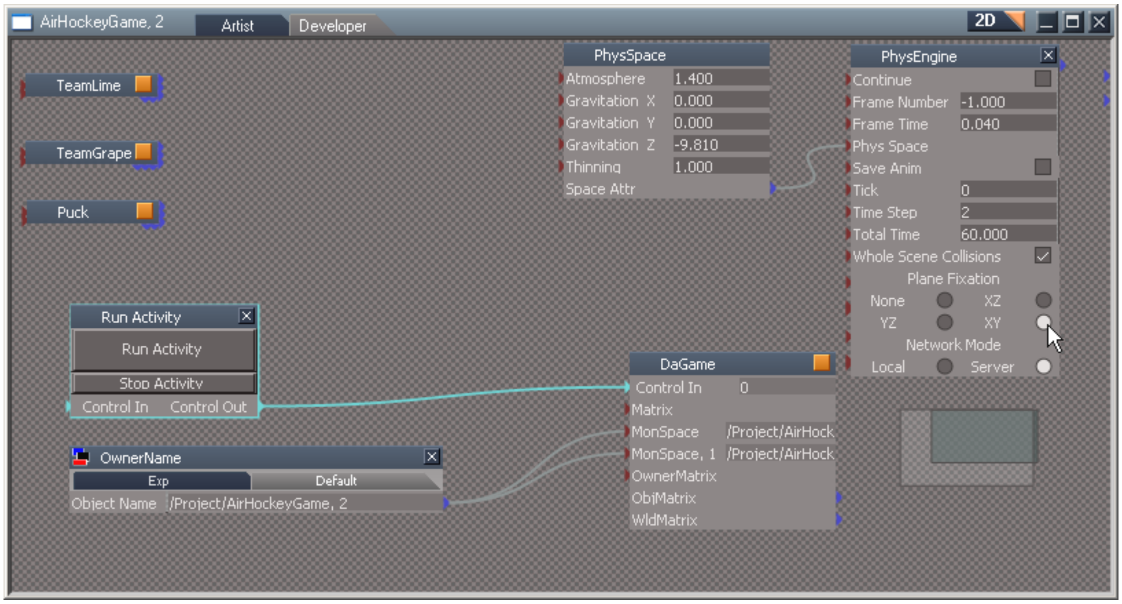
11.5.3 Tutorial: AirHockey Game

The AirHockey game was created as an example of physics within trueSpace.



AirHockey game

In a real air hockey game, you use paddles to deflect the puck across the surface of the ice area. The puck floats on a cushion of air and can move quite fast.



PhysSpace	
Atmosphere	1.400
Gravitation X	0.000
Gravitation Y	0.000
Gravitation Z	-9.810
Thinning	1.000
Space Attr	

PhysEngine	
Continue	☐
Frame Number	-1.000
Frame Time	0.040
Phys Space	
Save Anim	☐
Tick	0
Time Step	2
Total Time	60.000
Whole Scene Collisions	✓

Plane Fixation: None, XZ, YZ, [✓ XY]

Network Mode: Local, [✓ Server]

DaGame	
Control In	0
Matrix	
MonSpace	/Project/AirHock
MonSpace, 1	/Project/AirHock
OwnerMatrix	
ObjMatrix	
WldMatrix	

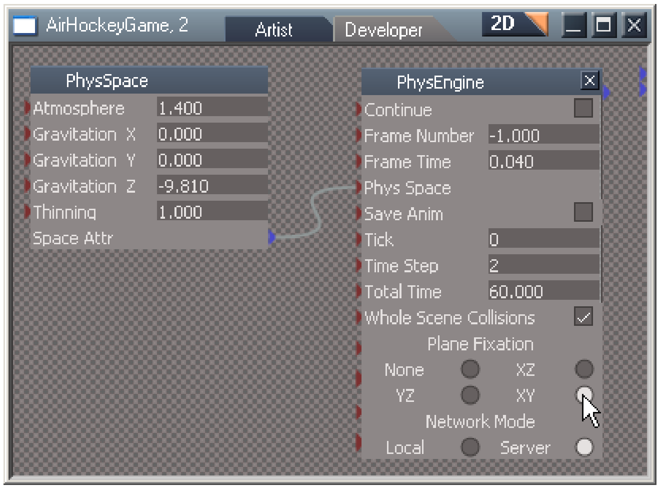
OwnerName	
Exp	Default
Object Name	/Project/AirHockeyGame, 2

Figure 1: AirHockey game in the LE.

The same principles exist in this virtual version of the game.

The PhysEngine object in trueSpace has the ability to limit or restrict which planes gravity will react on (XY, XZ, YZ, or none).

For this particular scenario, physics is restricted to the XY plane, which represents the ice surface in the game.



PhysSpace

Atmosphere	1.400
Gravitation X	0.000
Gravitation Y	0.000
Gravitation Z	-9.810
Thinning	1.000
Space Attr	

PhysEngine

Continue	☐
Frame Number	-1.000
Frame Time	0.040
Phys Space	
Save Anim	☐
Tick	0
Time Step	2
Total Time	60.000
Whole Scene Collisions	✓

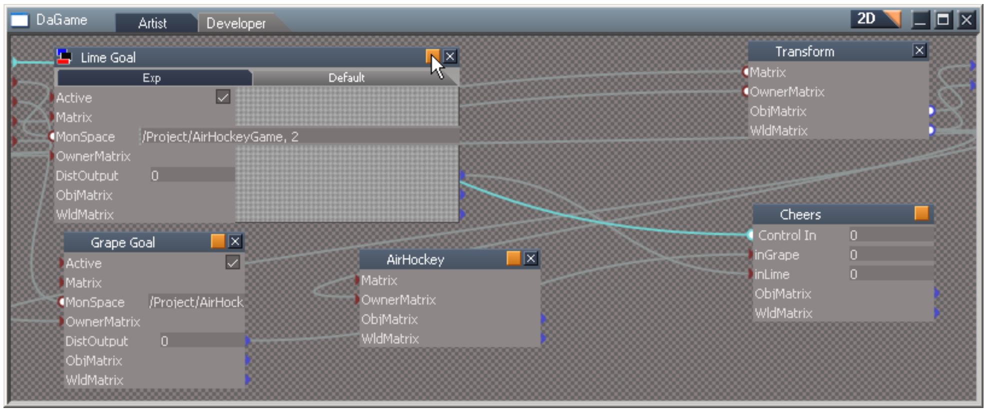
Plane Fixation: None, XZ, YZ, [✓ XY]

Network Mode: Local, [✓ Server]

Figure 2: Plane Fixation set for XY

The modeling portion of this scene is pretty straightforward. First, the ice surface and surrounding boards were created to test the plane fixation. The puck and team paddles are simple cylinder primitives. The only items with physical properties are the puck and the team paddles. Once the physical simulation is started, you use the Physics Move tool to move your paddle into the path of the puck. Upon collision, the puck moves in a new direction, hopefully towards the opposing team's goal line. Objects with Physics Attributes react to objects around them, whether those objects have physical attributes or not. The boards surrounding the rink serve to keep the puck on the playing surface. The goal areas allow the puck to enter and perhaps bounce back out into the playing surface area.

To add a little flavor into the scenario, remember the Fence scene, where a red light or a green light would turn on depending on the color of the sphere that moves towards it. This would be handy and allow a "goal-light" to turn on if the puck enters the net. The following images illustrate the inner workings of the AirHockey game:



Lime Goal
Exp | Default
Active ✓
Matrix
MonSpace /Project/AirHockeyGame, 2
OwnerMatrix
DistOutput 0
ObjMatrix
WldMatrix

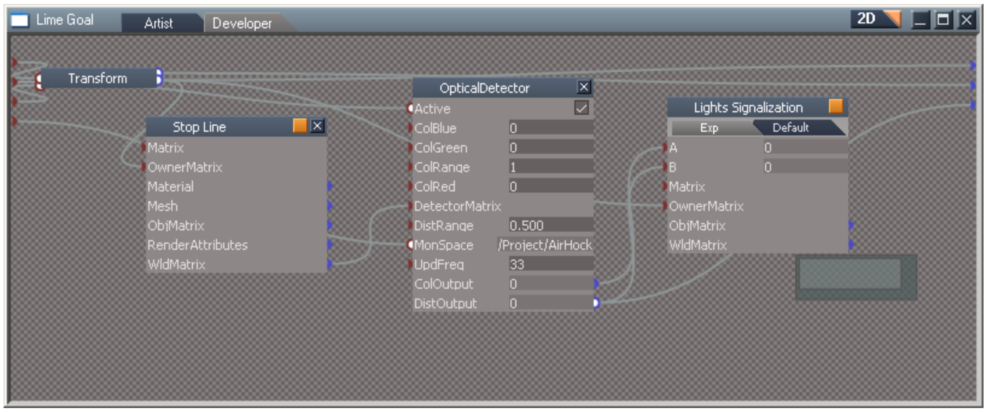
Transform
Matrix
OwnerMatrix
ObjMatrix
WldMatrix

Cheers
Control In 0
inGrape 0
inLime 0
ObjMatrix
WldMatrix

Grape Goal
Active ✓
Matrix
MonSpace /Project/AirHock
OwnerMatrix
DistOutput 0
ObjMatrix
WldMatrix

AirHockey
Matrix
OwnerMatrix
ObjMatrix
WldMatrix

Inside DaGame Object



Transform

Stop Line
Matrix
OwnerMatrix
Material
Mesh
ObjMatrix
RenderAttributes
WldMatrix

OpticalDetector
Active ✓
ColBlue 0
ColGreen 0
ColRange 1
ColRed 0
DetectorMatrix
DistRange 0.500
MonSpace /Project/AirHock
UpdFreq 33
ColOutput 0
DistOutput 0

Lights Signalization
Exp | Default
A 0
B 0
Matrix
OwnerMatrix
ObjMatrix
WldMatrix

Figure 3: Using existing objects to create complex scenarios

Of course with any complex activity, there is some customization which will happen. Here, using parts of the Fence activity’s “Secure” object, the Lime and Grape Goal objects use the “Stop Line” as a goal line. The Optical Detector and Lights Signalization had to be customized a little as well. The point is that you have some excellent examples that pre-exist within trueSpace, which can be utilized elsewhere.

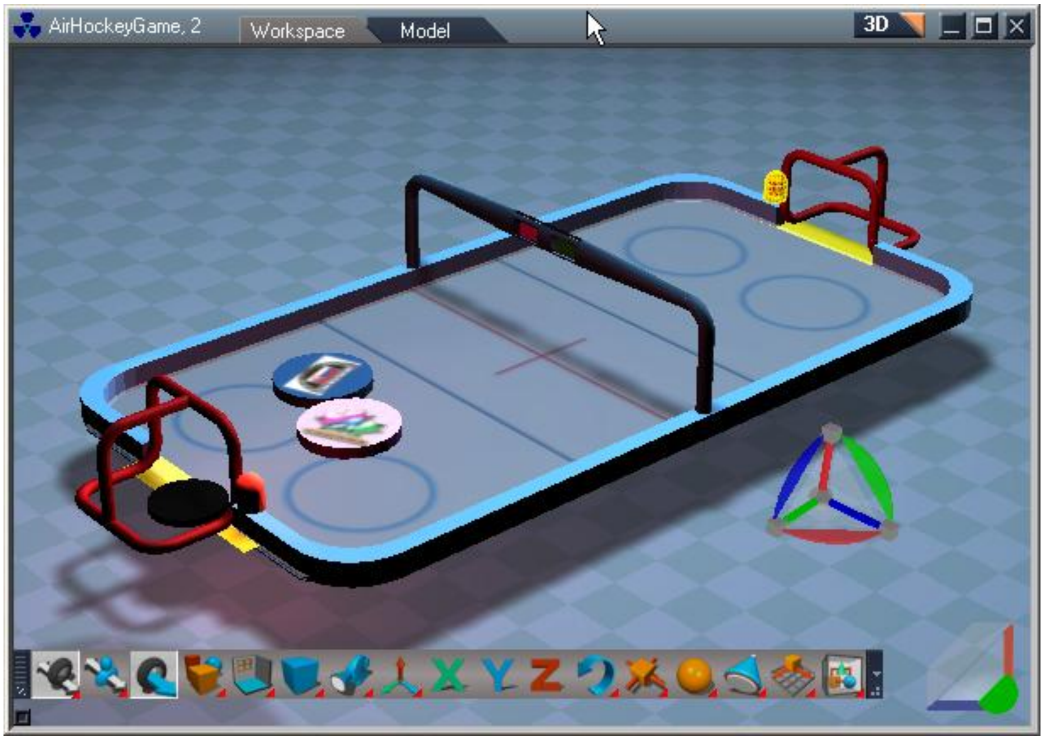
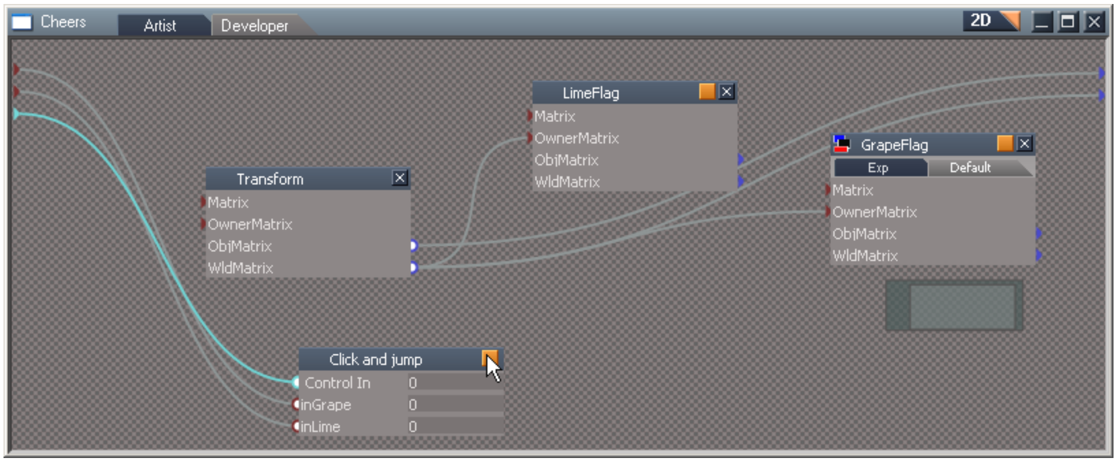


Figure 4: When the puck crosses the goal-line, the red light turns on.

Once the goal line scenario was completed, one additional activity was added to the scenario to demonstrate the use of pre-existing items in trueSpace.
There is an activity called Click'n'Jump, where objects jump when you click on them. The part of the scenario that causes object to jump was applied in a custom fashion to this game. In the first image of this tutorial (Figure 1), there is a "Run Activity" object. This starts or stops this next portion of the scenario.



Inside the cheers object

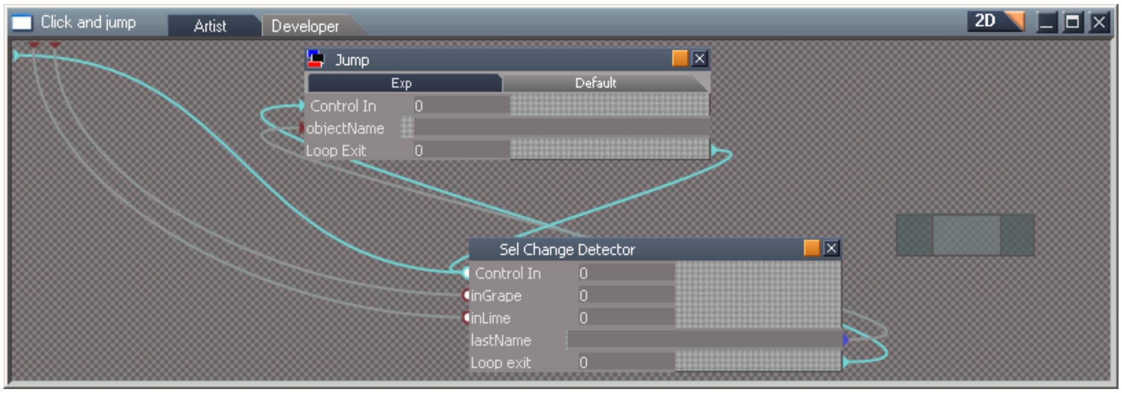


Figure 5: Using the Click'n'Jump activity in a custom fashion

Encapsulated inside the Cheers object you will find Click'n'Jump object. In short, either of the goal lines in the game will trigger respective colored flag to jump when a goal is scored. The ability to turn the flag jumping on or off just demonstrates additional control available within the game scenario. You can take the customization even further by finding (or creating) and adding a Score Board to this scenario.