Shared Worlds with Syzygy
assorted Shared World scripts running on a Phleet
Math 198

Andrew C Ofisher, Ryan T Mulligan

May 12, 2006

Abstract

By operating a Phleet among an arbitrary number of heterogeneous physical machines, one can use Syzygy scripts to create and manipulate a Shared World. Script demos explore different aspects of Scenegraph sharing. Detailed documentation of Phleet setup for various configurations is also provided. Included are the Phleet and Syzygy setups for Windows, Linux 32 bit, Linux 64 bit, Mac OS X, and Mac OS X x86 operating systems. Phleet and Syzygy instructions included for running a Phleet of Virtual Computers on a single physical computer.

Contents

List of Figures

1 Definitions

Virtual Environment The amalgamation of graphics, sound, and tactile outputs that make up a coherent experience.

Syzygy Syzygy is a computer program that allows you to build Virtual Environment computer applications.

Phleet Any number of Physical Computers and Virtual Computers being serviced by a single Syzygy Server.

Virtual Computer A single entity which abstracts one or more Physical Computers. A single Physical Computer can run an arbitrary number of Virtual Computers within itself, and belong to an arbitrary number of Virtual Computers spanning multiple Physical Computers.

Scenegraph A collection of information about how to execute a Virtual Environment.

Shared World The result of sharing a Syzygy Scenegraph among multiple Virtual Computers in the same Phleet [1]

Dex D(istributed)ex(ecution). This command runs a Syzygy program on a specific Virtual Computer.

EZSZG A unix shell that establishes many of the proper paths for running Syzygy commands and applications on a Physical Computer.

Batchfile A batchfile is a plain text file that initializes all of the paths and operating procedures of the Virtual Computers in a Phleet. Only Controller Computers need batch files.

SZG Daemon A daemon program that maintains a local SZG Database, and handles Dex requests.

Peer Computer A physical computer that is part of a Phleet and will be running a SZG Daemon.

[1] Though not a requirement, a Shared World typically results in both Virtual Computers displaying the same Virtual Environment.

Server Computer A physical computer that hosts a szgserver for a Phleet. There must be one and only one Server Computer per Phleet.

Controller Computer The computer on which someone executes Dex commands to the rest of the Phleet. An arbitrary number of Controller Computers can exist in a Phleet.

2 Outline for Creating a Shared World

1. Setup a Phleet (See section 3)
2. Create or acquire a shared–world enabled Syzygy script(s)(See section 4).
3. Dex the script(s) on each Virtual Computer in the Phleet

3 Setting Up A Phleet

Setting up a Phleet can be a complicated task, involving permissions, firewall and network issues. For this section it is assumed that you have managed to get all the physical computers in a state where they can communicate with each other.These instructions also assume that reader knows how to enter an EZSZG shell. None of the three Computer Instruction types are mutually exclusive of one another, for instance a single Physical Computer can be a Server, Peer, and Controller Computer.

3.1 Server Computer Instructions

1. Place a szg.conf file in its appropriate location. (See section 3.4)
2. In an EZSZG shell type

   ```
   szgserver ServerName Port &
   ```

   where “ServerName” is the arbitrary name of your server, and “Port” is the port number on which you want your server to communicate. The “&” indicates that the szgserver should run as a background process. Conventionally szgservers are named with the prefix “srv”.

   Examples:

```
szgserver srvlab 4343 &
szgserver srvisl 4343 &
szgserver srvhome 4343 &
```

3. Your server is now set up.

3.2 Peer Computer Instructions

1. Place a szg.conf file in its appropriate location. (See section 3.4)
2. In an EZSZG type

   ```
   dlogin ServerName YourName
   ```

   where "ServerName" is the what you called the server and "YourName" is your real first name, or any arbitrary identifier. If your server is not broadcasting its name properly you can alternatively dlogin using the server's IP Address by typing

   ```
   dlogin ServerIPAddress Port YourName
   ```

   Examples:

   ```
     dlogin srvlab ryan
     dlogin 111.111.111.111 4343 ryan
   ```

3. Run the SZG Daemon by typing

   ```
   szgd&
   ```

   in an EZSZG.

3.3 Controller Computer Instructions

1. Place a szg.conf file in its appropriate location. (See section 3.4)
2. In an EZSZG type

   ```
   dlogin ServerName YourName
   ```

where “ServerName” is the what you called the server and “YourName” is your real first name, or any arbitrary identifier. If your server is not broadcasting it’s name properly you can alternatively dlogin using the server’s IP Address by typing

```
dlogin ServerIPAddress Port YourName
```

Examples:

```
  dlogin srvlab ryan
  dlogin 111.111.111.111 4343 ryan
```

3. In an EZSZG shell type

   ```
   dbatch BatchfileName
   ```

   where BatchfileName the path to the plain text file which contains your Batchfile. (See section 3.5)

   Examples:

   ```
   dbatch debautch
   dbatch winprofile
   dbatch foo.txt
   ```

3.4 Syzygy Config Files

Syzygy Config files contain information specific to a Physical Computer. The placement of these files on the local filesystem is crucial. The default path locations can be found in Table 1. To override the default path location you need to set the SZG_CONF environment variable on your system[2]. By default the SZG_CONF environment variable is NULL. If you set it to a system path, Syzygy will look at that path for a szg.conf file.

A sample szg.conf file can be found in Figure 1. Many of the changes to this file are fairly self-explanatory. To edit them simply change the value between the tags. The name tag is the name of the Physical Computer where the file is located. The first interface should always have the name “internet” and it’s address tag should be the ip address of the Physical Computer. The

[2]On Unix, the default szg.conf path is in owned by root, this means to change anything with the command line szg.conf tools you need to use sudo, or define a non-default path for the szg.conf file in a user owned area.

Operating System	Path
Microsoft Windows	C:\szg\szg.conf
Unix Based	/etc/szg.conf

Table 1: default szg.conf paths
for various operating systems

mask should be the sub-net mask for this Physical Computer. The ports configuration should be left alone as the default, but should be there so that Syzygy knows what ports to use for communication.

The only tricky thing is the second interface listed in the file. This is the interface corresponding to the other Physical Computer in the Phleet. A szg.conf file needs to have an interface to every other Physical Computer in the Phleet if the Phleet traverses multiple sub-nets, or communicating across the internet[3] [4].

3.5 Batch Files

A batch file is an XML document. It specifies the workings of a Phleet in such a way that a Controller Computer can figure out what commands to send to the Syzygy Daemons running on the Peer Computers. The file begins with the

```
<szg_config>
```

tag and can include

```
<comment></comment> and <assign></assign>
```

tags. The file ends with the

```
</szg_config>
```

tag. The syntax for assign statements is the same as a when using the dset command.

[3]This means that the Phleet described by this szg.conf file has two computers named laptop and desktop. The laptop's IP address was 192.168.4.1 and the desktop's IP address was 192.168.5.1. Both were being connected to one another in a Shared World.

[4]This seems to violate the purpose of a szg.conf file because you are storing stuff about the Phleet in it when it seems that the original purpose of it was to store just information about particular Physical Computers and never require changing. Unfortunately this requirement has been verified by us when doing Syzygy experiments over the Internet.

```
<computer>
      <name>laptop</name>
    </computer>

    <interface>
      <type>IP</type>
      <name>internet</name>
      <address>192.168.4.1</address>
      <mask>255.255.248.0</mask>
    </interface>

    <interface>
      <type>IP</type>
      <name>desktop</name>
      <address>192.168.5.1</address>
      <mask>255.255.248.0</mask>
    </interface>

    <ports>
      <first>
      4700
      </first>
      <size>
      200
      </size>
    </ports>
```

Figure 1: szg.conf file

```
<comment>
Physical Computer Configuration
</comment>
<assign>
  desktop SZG_EXEC path c:\Python24;c:\szg-1.0\bin
  desktop SZG_PYTHON path c:\szg-1.0\doc\python
  laptop SZG_PYTHON path /usr/local/szg-1.0/doc/python
  laptop SZG_EXEC path /usr/bin;/usr/local/szg-1.0/bin
</assign>
```

Figure 2: Physical Computer Configuration part of a Batchfile "desktop" is running Microsoft Windows and "laptop" is running Debian Linux. (Note how linux paths still use ; as the seperator)

```
 ComputerName SZG_VariableSpace variable value
```

where "ComputerName" is the Physical Computer name defined in a szg.conf file3.4 or a name you have picked for a Virtual Computer, and SZG_VariableSpace is one of the variable spaces talked about below.

Physical Computer Configuration Statements Certain paths must be set so that the Controller Computer can tell the Physical Computers of the Phleet where to look for Syzygy files. These paths need to be set for every Physical Computer in the Phleet. Below is a list of the SZG_VariableSpaces applicable to Physical Computer Configurations.

SZG_EXEC This stores the path to any program that a Physical Computer depends on to run Syzygy. Examples include: szgrender, inputsimulator, python. This path need not include the path to Python based Syzygy scripts!

SZG_PYTHON This stores the path to any Python based Syzygy script that you wish your controller computer to be able to run.

Virtual Computer Configuration Statements Below is a list of the SZG_VariableSpaces applicable to Virtual Computer Configurations.

SZG_CONF This stores various configurations for the Virtual Computer. Variables associated with this Variable Space are listed below.

virtual A value of true tells the Syzygy Database to treat the "ComputerName" as a virtual computer.

location This is a local identifier name for Scene Graph Data. Keeping this the same as the Virtual Computer name makes sense in almost all applications.

relaunch_all A value of true means the Syzygy Daemon associated with the Virtual Computer will always restart all processes associated with the Virtual Computer when starting a new application. A value of false means the Syzygy Daemon will only restart processes that are dependant on the application and not already started.

SZG_TRIGGER This tells the Controller Computer which Physical Computer should be sent szgd requests, when the Virtual Computer is called to do something.[5] [6]

map Needs a value of a Physical Computer name.

SZG_DISPLAY This stores the display configurations for this Virtual Computer that are the same among all the displays.

number_screens Specifies the number of screens this Virtual Computer renders

SZG_DISPLAY# This stores the display configurations for a certain screen.

map Specifies a physical computer and a screen name on which to display part of the Virtual Environment.

networks Specifies the network on which the Controller Computer should look for the Physical Computer specified in map.

SZG_INPUT# This stores the input configurations for a certain input device.

map Specifies a physical computer and an input device name

networks Specifies the network on which the Controller Computer should look for the Physical Computer specified in map.

[5] In figure 3 the Virtual computer is only one computer, more complicated setups were beyond the scope of this paper.

[6] Additional information about more complicated trigger mappings can be found by looking at the Cave and Cube batch files directly.

```
<assign>
  vc SZG_CONF virtual true
  vc SZG_CONF location vc
  vc SZG_CONF relaunch_all false
  vc SZG_TRIGGER map desktop
  vc SZG_DISPLAY number_screens 1
  vc SZG_DISPLAY0 map desktop/SZG_DISPLAY0
  vc SZG_DISPLAY0 networks internet
  vc SZG_INPUT0 map desktop/inputsimulator
  vc SZG_INPUT0 networks internet

  vc2 SZG_CONF virtual true
  vc2 SZG_CONF location vc2
  vc2 SZG_CONF relaunch_all false
  vc2 SZG_TRIGGER map laptop
  vc2 SZG_DISPLAY number_screens 1
  vc2 SZG_DISPLAY0 map laptop/SZG_DISPLAY0
  vc2 SZG_DISPLAY0 networks internet
  vc2 SZG_INPUT0 map laptop/inputsimulator
  vc2 SZG_INPUT0 networks internet
</assign>
</szg_config>
```

Figure 3: Virtual Computer configuration partial Batchfile

4 Shared Worlds

A Shared World in Syzygy is implemented through node sharing between two or more different Scenegraphs. Shared Worlds allow a complete Scenegraph tree or parts of a Sceengraph tree to be shared between Virtual Computers. The sharing is accomplished by using the arGraphicsPeerc class. arGraphicsPeer allows for different sharing levels and also includes support for filtering by node level. See section 4.1 and 4.2 respectively.

4.1 Sharing

Shared worlds can be created using three different types of sharing. The types of sharing are defined by the type of peers that are used. There are feedback, push, and pull peers. There is also another subset of these peers in which geometry and transformations are not shared but tree structure is shared[1].

The first type of sharing, “feedback”, is the simultaneous sharing of two node trees. The node tree and updates to this tree are synchronized between the two peers. Updates are defined as changes to nodes such as changes to a transform matrix or color of a material. The second, “push”, is used when one peer pushes its node tree and updates to another peer but the remote peer does not send any of its own node tree nor any changes to its copy of the originating peers node tree. The third, “pull”, is exactly the reverse. A pull peer receives a node tree and updates from a remote peer but does not send its own node tree or any changes it makes.

The sharing type is defined by the pullSerial and pushSerial methods of the arGraphicsPeer class. Both functions take the following arguments: remote node name, remote node id, local node id, send level, remote send level, local send level. Remote node name is the name of the remote node that the link is to be made with. Remote node id is the node id of the remote node from which sharing should take place. Usually this is set to 0 to indicate the root node of the remote arGraphicsPeer. Local node id is the node id of the local node from which sharing should take place. Again, this is usually set to 0. Send level is the level at which nodes with a node level of equal or lower value will be shared. Remote send level is the level at which remote nodes with a node level of equal or lower value will be sent. Local send level is the level at which local nodes with a nodel level of equal to lower value will be sent. For a discussion on node levels see the section on filtering (Section 4.2). pushSerial is used to push the current peers node tree to another peer. pullSerial is used to pull a remote peers current node tree. Some examples:

From peer2.py:

```
peer.pullSerial("world0", 0, 0, AR_TRANSIENT_NODE, \
   AR_TRANSIENT_NODE, AR_TRANSIENT_NODE)
```

This command copies the current node tree from world0. Updates from the remote peer, world0, will be sent to this peer and updates this peer makes to the node tree will be sent to world0.

From buildworld.py:

```
pullSerial(sys.argv[i], 0, 0, AR_TRANSIENT_NODE, \
                AR_TRANSIENT_NODE, AR_IGNORE_NODE)
```

This command copies the current node tree from one of the worldpart counterparts. (See the documentation on buildworld and worldpart for more information.) Remote updates are sent to this peer but any changes that this peer makes to the scene graph will not be sent back.

4.2 Filtering

Filtering allows for a peer to select which node updates to send. For this to occur, nodes are assigned different levels: AR_TRANSIENT_NODE, AR_OPTIONAL_NODE, AR_STABLE_NODE, AR_STRUCTURE_NODE, AR_IGNORE_NODE. By default, new node is given the level AR_STRUCTURE_NODE. Filtering starts off by taking a node and transversing its tree. Any node below the given node with a node level less than or equal to the filtering level will still send updates. For a better example, see Figure 4. In the diagram, two peers are setup for feedback sharing via the nodes indicated by the arrows. Peer 2, however, is filtering from the root node at level 1, AR_STABLE_NODE. This means, the red node, if updated on Peer 2, will not change in Peer 1. The node below the red node, however, will have its updates sent from Peer 2 to Peer 1 and vis versa. Sharing is accomplished by the remoteFilterDataBelow and localFilterDataBelow methods of the arGraphicsPeer class. Both functions take three arguments: peer name, node id, node level. Peer name is the name of the remote peer. Node id is the id of the node from which filtering will take place. Node level is the level where node updates will be cut off. A node level of AR_OPTIONAL_NODE will only block nodes of level AR_TRANSIENT_NODE since it is the only level of greater value. For an example of filtering, see 4.4.

4.3 Running a Shared World on One Computer

In a typical Shared World setup with Syzygy, one or more Physical Computers are used for each Virtual Computer. For example, peer.py [7] is setup to run on two or more Virtual Computers, each of which make a sphere and share it with the others. In the course of testing and development Shared World scripts, multiple Physical Computer set ups are not always feasible. Fortunately, Syzygy supports multiple Virtual Computers on a single Physical Computer. This set up can be used to test Shared World scripts[8]

Initial Setup Before attempting to run any of the examples included using one Physical Computer, Syzygy must be configured and setup properly to run multiple Virtual Computers. The Physical Computer must be set up to be a Server, Peer, and Controller Computer[9]. See section 3 for more details. The steps to doing so are briefly outlined below[10]:

1. Configure a working szg.conf file
2. Start a szgserver
3. dlogin to the server
4. Run a szgd
5. dbatch singlecomputer.txt (Figure 5)

Running The next step is to load two copies of szgrender and inputsimulator, one for each Virtual Computer. In this example foo and bar will be used as the Virtual Computer names. From the EZSZG shell, enter the following:

```
dex szgrender -szg virtual=foo -szg mode/graphics=SZG_DISPLAY0
dex inputsimulator -szg virtual=foo
```

[7]peer.py is distributed with Syzygy in doc/python

[8]It has been observed that moving from one Physical Computer to an arbitrary number of them is trivial, given the Physical Computers are a properly set up Phleet

[9]The Controller Computer portion of the setup is done partly on the fly with use of the -szg flag. Your batchfile only needs to define paths, and not Virtual Computers. You can define the Virtual Computers in the batchfile if you want though, and thus not have to use the -szg flag.

[10]the steps are to all be performed on the same Physical Computer

```
dex szgrender -szg virtual=bar -szg mode/graphics=SZG_DISPLAY1
dex inputsimulator -szg virtual=bar
```

The dex commands use the -szg arguments to determine the Virtual Computer to run the component with respect to. The second -szg argument for the szgrenders is used to make sure each szgrender uses a different display window. If the display window name is not set Syzygy will issue an error stating that the current display slot is already in use. After these commands are issued, two szgrender windows should be displayed along with two inputsimulator windows[11].

Now we can dex scripts on the Virtual Computers. Here is the example syntax for the feedbackpeer. From the EZSZG shell, issue the following commands:

```
dex feedbackpeer1.py -szg virtual=foo
dex feedbackpeer2.py -szg virtual=bar
```

If all has gone well you will now see a sphere and cube in each of the szgrender windows (See section 4.4).

The dex commands use the -szg arguments to determine the Virtual Computer to run the component with respect to. The second -szg argument for the szgrenders is used to make sure each szgrender uses a different display window. If the display window name is not set Syzygy will issue an error stating that the current display slot is already in use. After these commands are issued, two szgrender windows should be displayed along with two inputsimulator windows[12].

Now we can dex scripts on the Virtual Computers. Here is the example syntax for the feedbackpeer. From the EZSZG shell, issue the following commands:

```
dex feedbackpeer1.py -szg virtual=foo
dex feedbackpeer2.py -szg virtual=bar
```

If all has gone well you will now see two spheres in each of the szgrender windows (See section 4.4).

[11]Warning: the second inputsimulator window will be in the exact location of the first making it seem like only one inputsimulator was launched.

[12]Warning: the second inputsimulator window will be in the exact location of the first making it seem like only one inputsimulator was launched.

4.4 Feedback Peers

feedbackpeer1.py and feedbackpeer2.py are very simple examples showing two peers which establish a feedback sharing mode and support filtering. feedbackpeer1 creats a sphere while feedbackpeer2 creates a cube. The button 1 changes the color of its object. Button 2 starts and stops bouncing of the local sphere. Button 3 enables filtering of the translation node for the opposite peer. Eg. if feedbackpeer1 enables filtering while the sphere created by feedbackpeer2 is bouncing, the sphere will freeze in its current location on feedbackpeer1 even though it remains bouncing on feedbackpeer2. While filtering is enabled, color changes to the sphere are still sent to feedbackpeer1. If filtering is then disabled, the sphere will once again resume bouncing on feedbackpeer1. See figure 6 for a screenshot.

4.5 Avatar Peers

avatarpeer1.py and avatarpeer2.py use feedback sharing one twist. avatarpeer1's Scenegraph contains a sphere and floor. The sphere changes colors via button 0 and will bounce via button 1 similar to feedbackpeer1. feedbackpeer2, however, is a little different. avatarpeer2 creates a blobbyMan avatar which follows the navigator. In other words, the blobbyMan follows the user as the user navigates around the environment. This movement is shared between the two peers so that movement in avatarpeer2 moves the avatar in avatarpeer1. This is behavior emulates the behavior in first person shooter games in which an avatar or characters movement is a function of the users location in the world. Figure 7 shows a screenshot.

4.6 Build World and World Part

buildworld.py and worldpart.py show how a shared world can be built up using parts from many independent peers who do not see or interact with each other. worldpart.py can be run on an arbitrary number of virtual computers. Each worldpart is completely independent and does not share or recieve any nodes from any other peer. buildworld.py assembles the contents of all the worldpart peers it is told about, and places them in one shared world. Running buildworld and worldpart is different than the other two examples. worldpart takes two arguments. The first is the name given to identify itself. The second is one of hexahedron, sphere, or plane. buildworld is passed the name of all the worldparts that it should assemble. Here is an example of 3 worldparts and 1 buildworld. The worldparts are assembled on virtual computers vc1, vc2, and vc3. buildworld is run on the virtual computer vc4. In each of the worldparts, navigation is tied to the object just like avatarpeer. Therefore, when an object is moved in any one of its worldpart environments, the movement can be seen in the buildworld environment.

```
dex vc1 worldpart.py part1 sphere
dex vc2 worldpart.py part2 hexahedron
dex vc3 worldpart.py part3 plane
dex vc4 buildworld.py part1 part2 part3
```

Figure 8 shows the resulting image on vc4 after some navigation and color change.

5 Syzygy Setup and Configuration

5.1 Compiling on Linux 64 bit and Mac OS X x86

64bit linux and Mac OS X on Intel require changes to the Syzygy source code in order to compile and run. These changes are mainly due to assumptions about pointers being 32bits and changes in GCC. The altered source code with the exact changes can be found at http://new.math.uiuc.edu/ ivanhoe/mulligan/src/syzygy/index.html

6 Conclusions and Future Improvements

6.1 Conclusions

Shared Worlds with Syzygy provides the basics for starting a Shared World project in a Virtual Environment.

6.2 Future Improvements

Syzygy has a great potential for building interesting Shared World applications. People should be able to use *Shared Worlds with Syzygy* as a stepping stone to projects of more great significance. A few notable examples would be video conferencing between Virtual Environments, psychological experiments with two subjects interacting with each other, and dynamical system viewers with each user being able to point at locations of interest.

It does seem as though it might be possible to teach a course in computer graphics/virtual environments using a Shared World where students and teachers use a client to connect to the world and then use python interactive bound to helpful methods to learn about transformations, lighting, and animation. It may make sense to use VPython as part of such a system as a starting place for placing objects into the shared world, and use python methods bound to Threading objects to facilitate animations through Python interactive without using an infinite loop. For instance the program would start by connecting to the Shared World, then it would start a Thread for the display and navigation devices of the client. After the program would drop into a Python Interactive session, where the user could type things like

```
>>>s = sphere(pos=(0,0,0), color=color.red)
```

and then call another Threading Object called Bounce something like

```
>>>b = Bounce(s) \ b.start()
```

If setup properly Bounce would cause the sphere to animate in some way in the Virtual Environment. Of course, the Teacher could then just look around inside the Virtual Environment from his computer and see the student's avatars and the sphere's that they had created and watch them bounce.

7 Special Notes

7.1 Input Simulator vs. the Cave and Cube

The inputsimulator and the actual wand in the Cave and Cube are two completely different ways to generate input to a Syzygy application. When using the inputsimulator, buttons will only work in mode 4 (Translate Wand + Buttons). There are now 4 different sections of buttons, 01, 23, 45, and 67. Different modes are selected by hiting the space bar and ar indicated by the white dot next to the red buttons in the inputsimulator. When in any mode, the left mouse button activates the first button (button 0 in mode 01) and the right mouse button (Apple + Left Click on Macintosh) activates the second button (button 1 in mode 01). In the Cave or Cube, buttons 0, 1, and 2 are the buttons on the bottom right of the controller in that order. These were previously known is left, middle, and right.

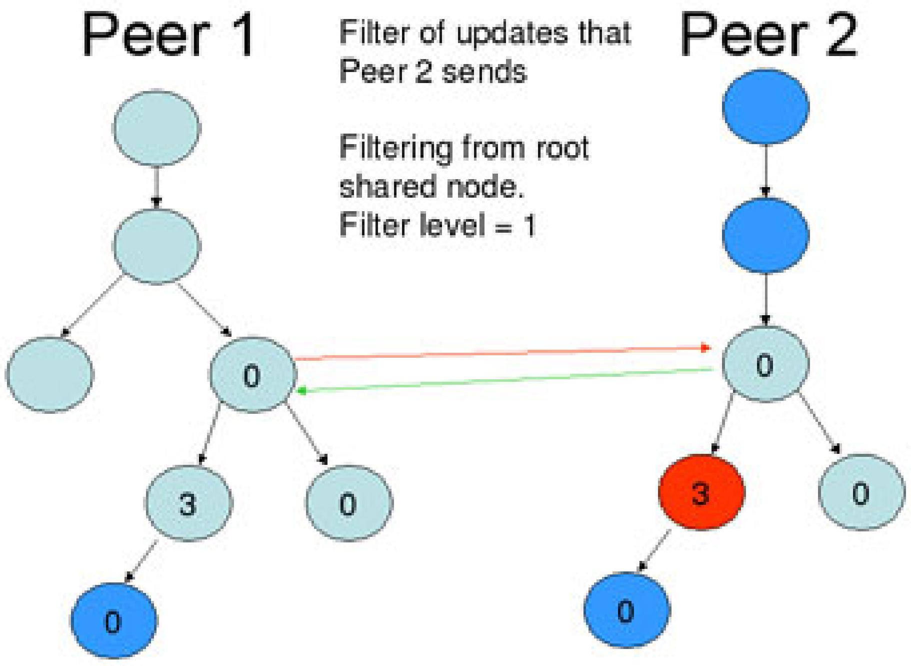
Peer 1
Filter of updates that Peer 2 sends
Filtering from root shared node.
Filter level = 1
Peer 2
0
3
0
0
0
3
0
0

Figure 4: Visualization of filtering

```
<szg_config>
  <assign>
    YourComputerName SZG_EXEC path c:\Python24;c:\szg-1.0\bin
    YourComputerName SZG_PYTHON path c:\szg-1.0\doc\python
  </assign>
</szg_config
```

Figure 5: singlecomputer.txt
while the example shows the setup is for Windows, a similiar one can easily be made for Unix

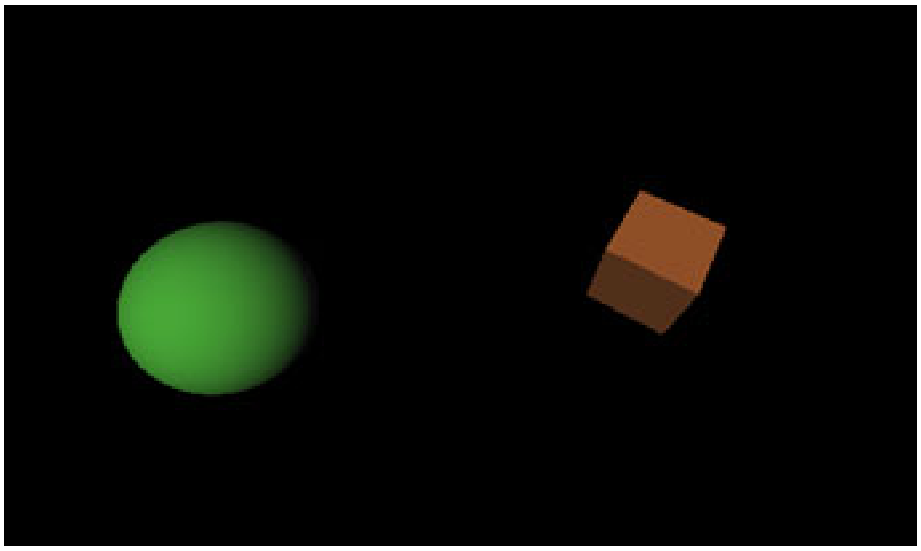

Figure 6: Screenshot of feedbackpeer1.py.

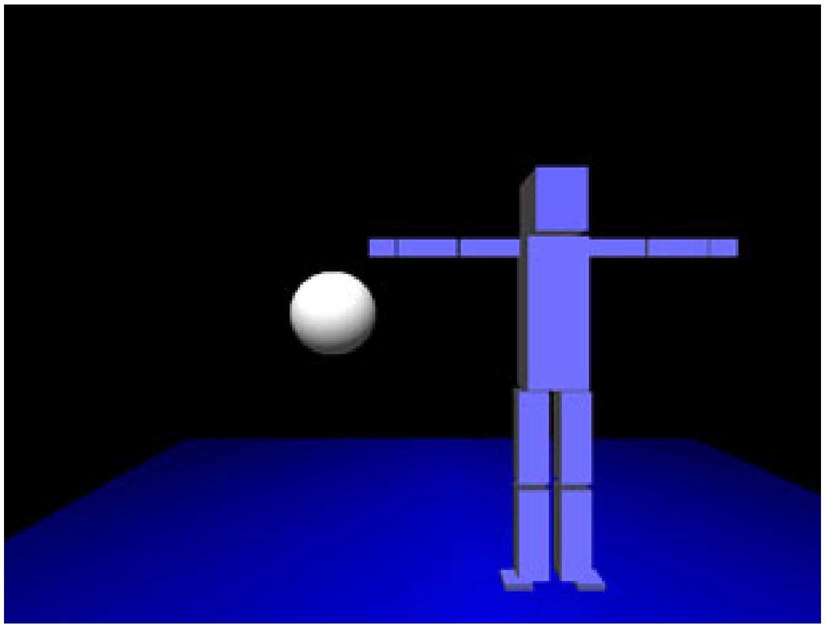

Figure 7: Screenshot of avatarpeer1.py.

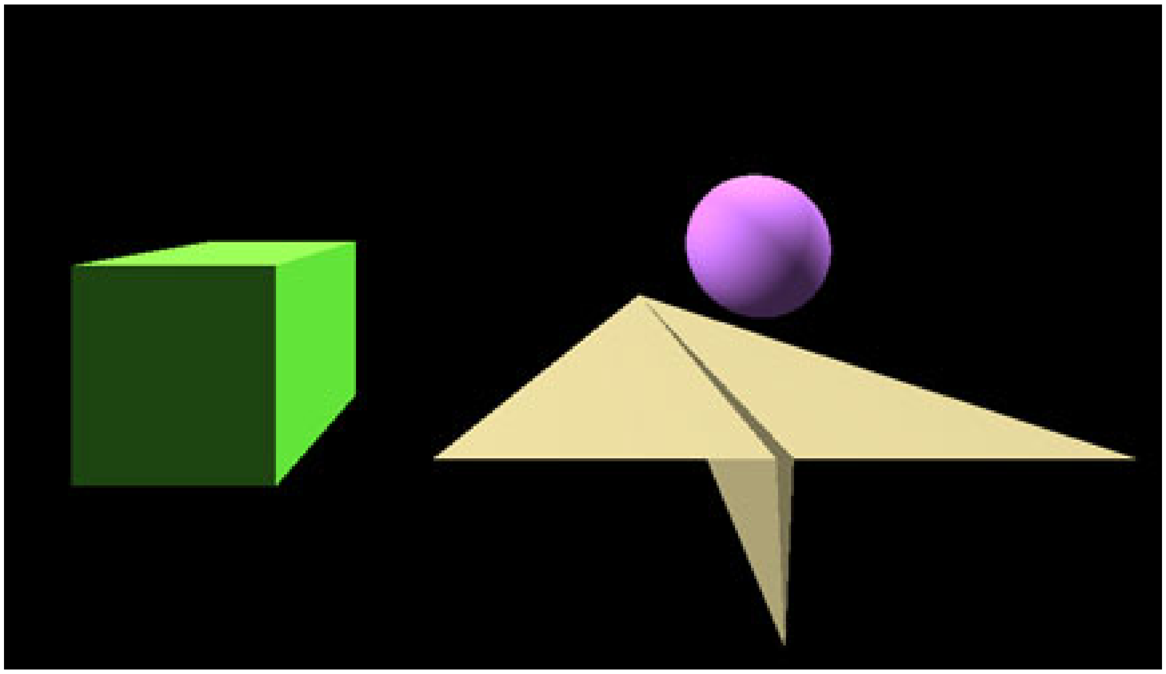

Figure 8: buildworld.py running with three peers.

References

[1] Ben Schaeffer, et. al.: *Myriad:scalable VR via peer-to-peer connectivity, PC clustering, and transient inconsistency*, http://www.isl.uiuc.edu/ schaeffr/vrst31-schaeffer.pdf

[2] Integrated Systems Lab: *Syzygy 1.0 Documentation* http://www.isl.uiuc.edu/szg/doc/index.html