PyView Version 1.0

Peter Brinkman and Brett Witt
IlliMath04 - August 2004

Abstract

PyView is an OpenGL application capable of displaying triangle meshes using the Szgygy distributed graphics framework. It is written completely in Python for flexibility and simplicity. As of this writting, it is capable of displaying binary OFF files generated using the Evolver software package. However, modules exist to allow PyView to display MD2 meshes as well as ASCII OFF files. Currently animation is only supported in the MD2 and binary OFF formats.

PyView is essentially embedded within the PyCrust engine itself. PyCrust is a graphics framework developed to simplify the creation of Python programs using the Szgygy Python bindings. However, it is important to note that both PyCrust and PyView are depreciated in favor of another framework currently in development called Flare.

1 Design

PyView was designed to be easily extensible through inheritence from the Model object interface. The interface defines only two functions, a Load function and an Export function. The Load function takes a file name and then parses through that file extracting data for position, color, normals, and face indexing. The export function returns a tuple containing a vertex list and a face list. Both of these lists are two-dimensional arrays, with the first dimension being the current frame, and the second dimension being the current vertex/face one is attempting to retrieve for that given frame.

The vertex list and the face list can then be supplied to the PyCrust Mesh class via a member function called GenMeshFromModel. This function takes the data supplied from export, calculates missing data, such as per vertex or per face normals, coloring, and a face list. This gives the user the ability to define as little data as possible and still get an object to display on screen. They can then later refine the object's parameters in the Load function to more accurately visualize a particular surface.

Navigation of an object in the world is mostly accomplished through a default navigator known as the Pape navigator. This allows a user to instantly visualize a mesh after having written a loader for it from every angle. From there the navigation interface is completely exposed to allow the user to define their own navigation system. This can be accomplished by modifying the navigation functionality SzyDisplay located in the Render function.

Surfaces can also be generated directly in code. Simply ignore the filename parameter of the Load function when inheriting an object from Model and algorithmically generate your data there. Then have the Export function output that data in the format specified above and in the documentation string for that function, and it will seamlessly integrate with the PyCrust Mesh class.

For extremely fine control over all aspects of the visualization of a surface, it is possible to completely circumvent PyView's Mesh class, although this process also circumvents the process through which PyView's rendering speed is comparable to that of some C++ implimentations. This can be accomplished by inheriting off the PyCrust RenderObject class and rendering your object manually using OpenGL calls. However it should be noted that without significant experience in OpenGL, rendering an object via this method could reduce your frame rate below what is acceptable for animation.

2 Usage

PyView and PyCrust are depreciated! Please use Flare instead!

While PyView should not be used for any project since it is being replaced with the much faster hybrid C++/Python/Ruby framework called Flare, this documentation is provided for educational and historical purposes.

Start by checking to see if there is support for the file format you are attempting to use. PyView supports binary OFF files, ASCII OFF files, and MD2 models. A bit of work is required to get the ASCII OFF files and the MD2 model files to work again, primarily because near the end of development those model loaders weren't modified to work with the current system. However if that support is required and not present in Flare, looking at the binary OFF model loader should be sufficient enough to make the required changes. Keep in mind that the mesh loader also unconditionally mirrors the data accross the xz axis so additional functions and parameters would need to be added in order to turn that feature off for things like the MD2 model format.

The MD2 file loader only loads in the keyframes currently. PyView cannot interpolate between keyframes because doing so would be unacceptably slow, this is one of the reasons for the creation of Flare. As such if animation is choppy when displaying the MD2 object, consider interoplating in the model loader and supplying those additional frames in your face list. This will allow the Mesh class to cache the data on the video card and should also provide for a much smoother animation at the expense of a longer load time.

Currently PyView also expects that vertex data is unique for each frame. While this is extremely odd in the field of real-time computer graphics, this default functionality needed to be provided since the OFF file format does not gureentee that the number of verticies stays consistant between two frames. If the model you are loading only has one set of vertex data, then simply make the other elements in the vertex list references of that single set of vertex data. This operation is trivial in Python.

For an example of how to write a file loader for a Mesh object, please examine BOffModel.py. This is by far the simplest file loader there is in the default distribution of PyView.

3 The Future

PyView was a simple proof of concept written completely in Python, and is fairly good at what its primary goal is, which is fast and easy visualization of triangle meshes in Python. However it is very limited, partially because Python is not descriptive enough to allow for an elegant object oriented framework.

Python is also not fast enough to justify the graphical back end being written in it when the code has one to one correspondence with equivalent C code since the OpenGL bindings do not take advantage of any of the language features of Python.

Flare address all of these concerns without sacrificing any flexibility on the part of the user. The complete rendering system is written in C++ and has far more functionality than PyView, such as vertex/pixel shader support, the ability to interpolate between key frames, vertex buffer object usage, and many other features. Another key point is that most of this functionality is enabled by default, almost instantly enhancing any program ported to use the Flare framework without sacrificing any speed and more often than not, experiencing a significant speed increase.

During the program initialization, render loop, and shutdown processes callback files are provided for writing additional instructions to be executed during those stages. They are tentatively named PyFlareInit, PyFlareLoop, PyFlareShutdown, RbFlareInit, RbFlareLoop, and RbFlareShutdown for Python and Ruby respectively. Since C++, Python, and Ruby all share the same address space, it is possible to write in the language that best suites a particular task. However currently for each stage you can only choose one language to write in, since the order in which each externally created file is processed is arbitrary.

Communication between the external languages and the C++ framework will take place using bindings that take advantage of the features of the particular language it's being ported to. A straight port of the classes and member functions of the C++ framework will be provided as well, however using those straight bindings will be unnessecary because the bindings written for that language will provide access to all the functionality of the underlying C++ API, just in a way that plays on the strengths of the language the bindings are being written for.

Automatic resource management allows users of Flare to use shaders, textures, and even instances of objects freely without worrying about duplication of data. The copy on write policy the resource manager defines ensures that resource management is completely hidden from the user.

Automatic synchronization ensures that clocks defined within your program are synchronized across all render nodes to ensure that your program will function in a distributed environment the same as in a stand-alone environment.

An advanced logging system has been built into Flare in an attempt to assist in the debugging of programs written using the Flare framework. This should cut down on untracable errors and downtime and increase productivity, especially when these errors occur in the Cube.

Additional functionality will be added to Flare over time to better address wants and needs of the researchers and students using it. Also since the rendering interface of the Flare framework is heavily abstracted, the Flare framework can be easily ported to any system using any 3D graphics API.