SGCT – Simple Graphics Cluster Toolkit

Introduction

SGCT is a static C++ library for developing OpenGL applications that are synchronized across a cluster of image generating computers (IGs). SGCT is designed to be as simple as possible for the developer and is well suited for rapid prototyping of immersive virtual reality (VR) applications. SGCT supports a variety of stereoscopic formats such as active quad buffered stereoscopy, passive side-by-side stereoscopy, passive over-and-under stereoscopy, checkerboard/DLP/pixel interlaced stereoscopy and anaglyphic stereoscopy. SGCT applications are scalable and use a XML configuration file where all cluster nodes (IGs) and their properties are specified. Therefore there is no need of recompiling an application for different VR setups. SGCT does also support running cluster applications on a single computer for testing purposes. SGCT is based on GLFW and GLEW.

Features

- Frame synchronization, swap buffer synchronization (hardware) and application data synchronization.
- PNG texture management
- GLSL shader management
- Freetype 2 font management
- Error messaging across the cluster
- Statistics and performance graphs
- External TCP control interface (telnet, GUI applications, Android apps, IOS apps)
- Easy thread handling (GLFW)
- Joystick and gamepad support (GLFW)

Limitations

Windows only at the moment (Visual Studio and MinGW) but will support more platforms in the future. Currently there is no tracking support in the library but VRPN support will be implemented in the future.

Getting started

XML configuration file

In order to run any application a valid XML configuration file that describes the VR setup must be provided. The simplest form of a cluster consists of a single node. Here is an example on how such configuration can look like.

```
<?xml version="1.0" ?>
<Cluster masterAddress="127.0.0.1">
     <Node ip="127.0.0.1" port="20401">
          <Window fullscreen="false">
               <Stereo type="none" />
               <Pos x="200" y="300" />
               <!-- 16:9 aspect ratio -->
               <Size x="640" y="360" />
          </Window>
          <Viewport>
               <Pos x="0.0" y="0.0" />
               <Size x="1.0" y="1.0" />
               <Viewplane>
                    <!-- Lower left -->
                    <Pos x="-1.778" y="-1.0" z="0.0" />
                    <!-- Upper left -->
                    <Pos x="-1.778" y="1.0" z="0.0" />
                    <!-- Upper right -->
                    <Pos x="1.778" y="1.0" z="0.0" />
               </Viewplane>
          </Viewport>
     </Node>
     <User eyeSeparation="0.069">
          <Pos x="0.0" y="0.0" z="4.0" />
     </User>
</Cluster>
```

In most 3D graphics applications a camera is defined by a vertical field of view (VFOV) and an aspect ratio. In VR you usually set up a view plane per viewport which corresponds to the physical dimensions of a screen (or channel if the screen is multi-channel). By doing so enables the use of unsymmetrical frustums which are needed for correct stereoscopy. The coordinates of the view plane and the user position are in meters and this result in that the OpenGL coordinates are in meters too. Note that the coordinate system origin can be shifted by offsetting the coordinates for the view plane and the user. In this case the origin is set to the middle of the screen.

SGCT supports only one window per node but can contain several viewports. The viewport coordinates are in normalized screen space [0.0, 1.0]. Each computer will compare its ip addresses with all nodes and the master address to determine which node configuration to use and if it's the master or not. When running several nodes on a single computer this process can be overridden by using command line arguments. This will be described in section XX.

First application

Creating applications using SGCT is very simple and requires minimum coding. Here is an example which just opens up a window and initiates the synchronization across the cluster.

```
#include "sgct.h"

sgct::Engine * gEngine;

int main( int argc, char* argv[] )
{
	gEngine = new sgct::Engine( argc, argv );

	if( !gEngine->init() )
	{
		delete gEngine;
		return EXIT_FAILURE;
	}

	// Main loop
	gEngine->render();

	// Clean up
	delete gEngine;

	// Exit program
	exit( EXIT_SUCCESS );
}
```

Let's draw a triangle. Create a draw function that doesn't take any arguments and doesn't return anything. Don't forget to declare the function.

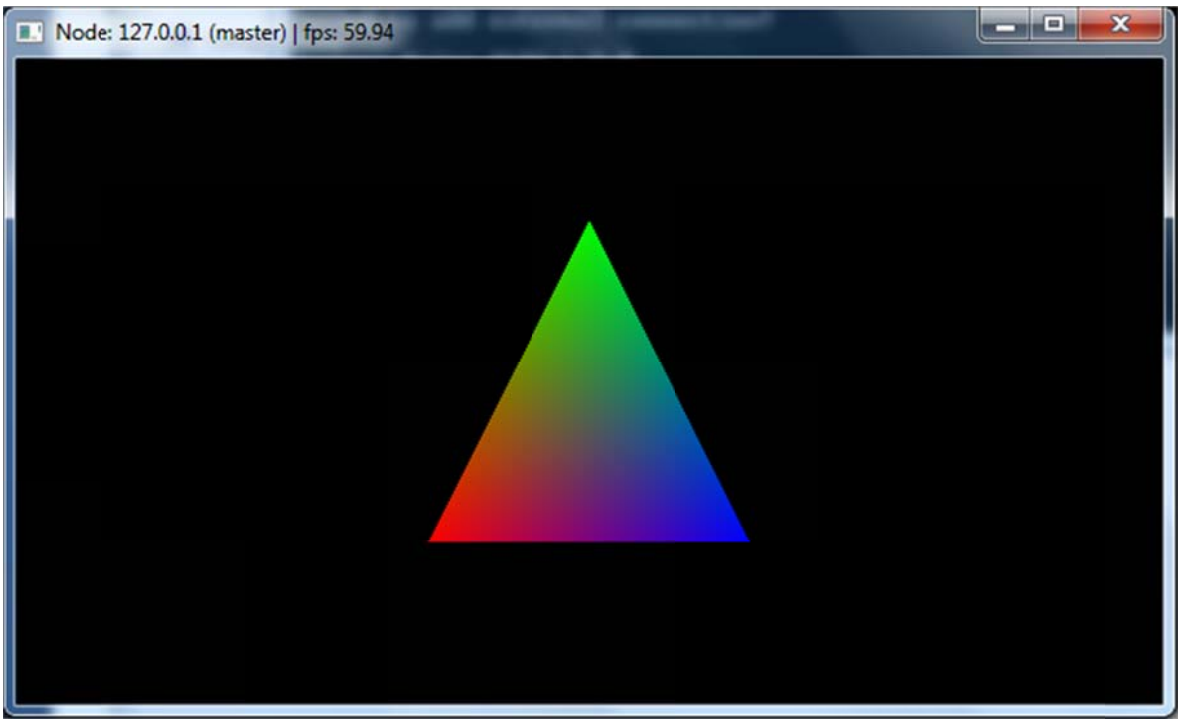
```
void myDrawFun()
{
	//render a single triangle
	glBegin(GL_TRIANGLES);
		glColor3f(1.0f, 0.0f, 0.0f); //Red
		glVertex3f(-0.5f, -0.5f, 0.0f);

		glColor3f(0.0f, 1.0f, 0.0f); //Green
		glVertex3f(0.0f, 0.5f, 0.0f);

		glColor3f(0.0f, 0.0f, 1.0f); //Blue
		glVertex3f(0.5f, -0.5f, 0.0f);
	glEnd();
}
```

Now we need to tell SGCT to use the render function.

```
//Bind your draw function to the render loop
gEngine->setDrawFunction( myDrawFun );
```

This call can be placed anywhere between where the engine object is constructed and where the engine starts the render loop. Compile and start the application with the following arguments: `-config "path_to_the_configfile.xml"`

It's possible to set the arguments in the project's options in your IDE (Integrated Development Environment). Now the result should look something like the image below.



Let's make the triangle spin, synchronized across a cluster. In order to do that a variable needs to be shared between the nodes. We can use the time from the Master node and send it to all the slave nodes. The shared variables are synchronized before the draw function is called in the pre draw function. Variables sent across the network needs to be serialized, the master encodes the variables and the slaves decodes them. Since it's a serial process the order of the variables needs to be the same in the decoder as in the encoder. The code below show how to synchronize the time variable:

```
void myPreDrawFun();
void myEncodeFun();
void myDecodeFun();

double time = 0.0;

void myEncodeFun()
{
	sgct::SharedData::Instance()->writeDouble( time );
}

void myDecodeFun()
{
	time = sgct::SharedData::Instance()->readDouble();
}

void myPreDrawFun()
{
	if( gEngine->isMaster() )
	{
		time = glfwGetTime();
	}
}
```

Set/bind the functions to SGCT by:

```
gEngine->setPreDrawFunction( myPreDrawFun );
sgct::SharedData::Instance()->setEncodeFunction(myEncodeFun);
sgct::SharedData::Instance()->setDecodeFunction(myDecodeFun);
```

Use the time variable to rotate the triangle around the Y-axis in your draw function before you draw the triangle:

```
float speed = 50.0f;
glRotatef(static_cast<float>( time ) * speed, 0.0f, 1.0f, 0.0f);
```

Download the source code and project files here: www…..