How to Develop Virtual Reality Applications in zSpace – Part II

infinite z

z space

Agenda

- Review
- System Requirements
- Enabling Stereo
- Head Tracking
- Stylus Tracking
- Direct Interaction
- Questions

z space®

Review

Direct interaction with 3D virtual-holographic simulations in open space

Full color, high resolution stereoscopic display

Passive, polarized tracking eyewear

Unique stylus for 3D interaction

Innovative software development platform

z space®

System Requirements

- Windows 7 or XP, 32bit or 64bit
- Microsoft Visual Studio 2008 SP1, 2010
- Stereo Enabled GPU
  - Pro only (nVidia Quadro or AMD FirePro)
- SDK is a collection of C++ classes
  - Simplified C interface in 2.7

z space®

Application Steps

- Stereo Enabled Rendering
- Use zSpace Stereo Data
- Enable Head Tracking
- Read Stylus
- Direct Interaction

Enabling Stereo

Setting The Pixel Format

```
// Obtain a device context for the window
m_hDc = GetDC(m_hWnd);

// Set an appropriate pixel format
PIXELFORMATDESCRIPTOR pfd = {
    sizeof(PIXELFORMATDESCRIPTOR),  1,
    PFD_DRAW_TO_WINDOW |  PFD_SUPPORT_OPENGL | PFD_STEREO | PFD_DOUBLEBUFFER,
    PFD_TYPE_RGBA,
    24, 0, 0, 0, 0, 0, 0,  0,  0,  0, 0, 0, 0, 0, 32, 0,  0,  PFD_MAIN_PLANE,  0,  0, 0, 0
};

int  pixelFormat;

// Get the best available match of pixel format for the device context
pixelFormat = ChoosePixelFormat(m_hDc, &pfd);

// Make that the pixel format of the device context
SetPixelFormat(m_hDc, pixelFormat, &pfd);
```

z space

Enabling Stereo

Setting Up the zSpace Stereo Objects

```
// Create the stereo window.
m_stereoWindow = new StereoWindow(windowX, windowY, windowWidth, windowHeight);

// Do left/right frame detect
StereoLeftRightDetect::initialize(m_stereoWindow, StereoLeftRightDetect::WINDOW_TYPE_GL);

// Create a viewport the size of the window
StereoViewport* stereoViewport = new StereoViewport(0, 0, windowWidth, windowHeight);

// Add the viewport to the stereo window
m_stereoWindow->addStereoViewport(stereoViewport);

// Have the viewport track stereo window size changes
stereoViewport->setUsingWindowSize(true);

// Remove the reference
stereoViewport->removeReference();
```

z space®

Enabling Stereo

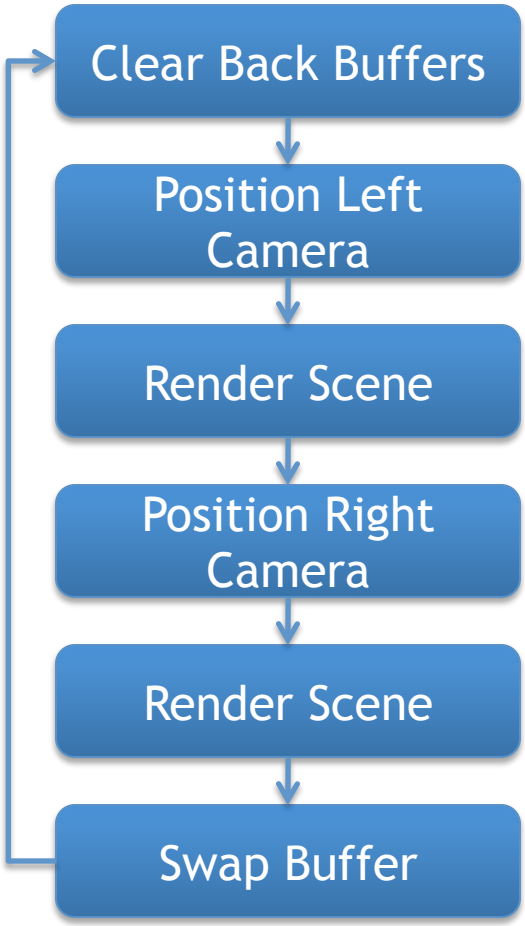
Setting Up the zSpace Stereo Objects

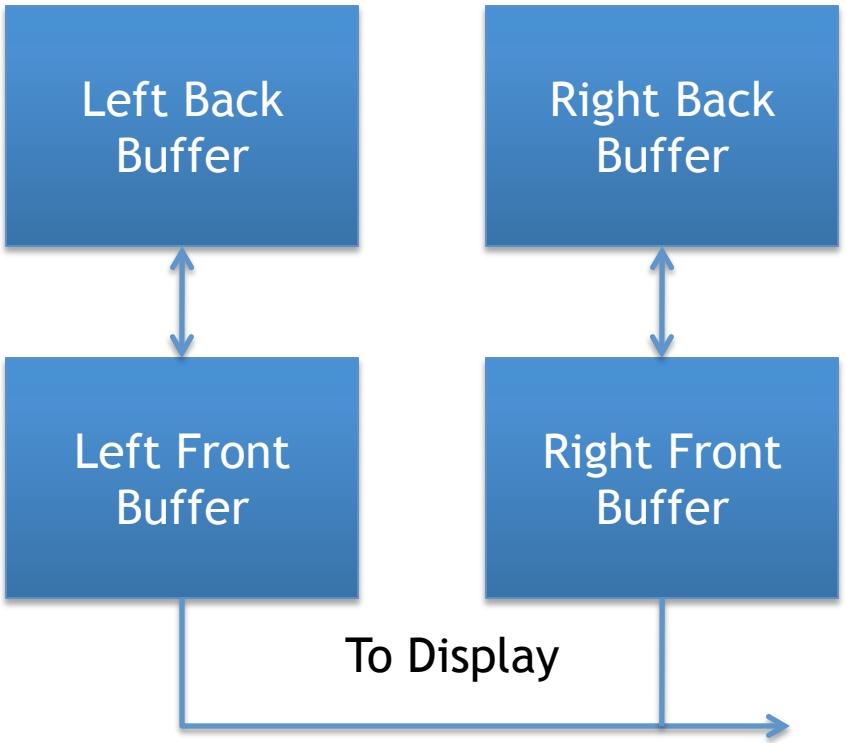
```
// Create the stereo window.
m_stereoWindow = new StereoWindow(windowX, windowY, windowWidth, windowHeight);

// Do left/right frame detect
StereoLeftRightDetect::initialize(m_stereoWindow, StereoLeftRightDetect::WINDOW_TYPE_GL);

// Create a viewport the size of the window
StereoViewport* stereoViewport = new StereoViewport(0, 0, windowWidth, windowHeight);

// Add the viewport to the stereo window
m_stereoWindow->addStereoViewport(stereoViewport);

// Have the viewport track stereo window size changes
stereoViewport->setUsingWindowSize(true);

// Remove the reference
stereoViewport->removeReference();
```

z space®

Enabling Stereo

Stereo 3D Rendering





z space

Enabling Stereo

The Stereo Render Loop

```
// The main render loop
while (m_done == false)
{
  renderEye(StereoFrustum::STEREO_EYE_LEFT);
  renderEye(StereoFrustum::STEREO_EYE_RIGHT);

  SwapBuffers(m_hDc);
}

// This happens for each eye
void renderEye(StereoFrustum::StereoEye eye)
{
  setupMatrices(eye);
  if (eye == StereoFrustum::STEREO_EYE_LEFT)
    glDrawBuffer(GL_BACK_LEFT);
  if (eye == StereoFrustum::STEREO_EYE_RIGHT)
    glDrawBuffer(GL_BACK_RIGHT);

  // Finally draw the scene
  drawScene();
}
```

Enabling Stereo

The Stereo Render Loop

```
// The main render loop
while (m_done == false)
{
  renderEye(StereoFrustum::STEREO_EYE_LEFT);
  renderEye(StereoFrustum::STEREO_EYE_RIGHT);

  SwapBuffers(m_hDc);
}

// This happens for each eye
void renderEye(StereoFrustum::StereoEye eye)
{
  setupMatrices(eye);
  if (eye == StereoFrustum::STEREO_EYE_LEFT)
    glDrawBuffer(GL_BACK_LEFT);
  if (eye == StereoFrustum::STEREO_EYE_RIGHT)
    glDrawBuffer(GL_BACK_RIGHT);

  // Finally draw the scene
  drawScene();
}
```

Application Steps

- Stereo Enabled Rendering
- Use zSpace Stereo Data
- Enable Head Tracking
- Read Stylus
- Direct Interaction

Enabling Stereo

Adjusting The Left and Right Cameras

```
GLfloat monoModelViewGl[16];
GLfloat viewMatrixGl[16];

// Copy the mono model-view matrix.
glMatrixMode(GL_MODELVIEW);
glGetFloatv(GL_MODELVIEW_MATRIX, monoModelViewGl);

// Get the stereo frustum.
StereoFrustum* stereoFrustum = m_stereoViewport->getStereoFrustum();

// Modify the model-view matrix for eye.
// Note: The view matrix contains eye offset plus any rotation required for off-axis projection.
Matrix4 viewMatrix4;
stereoFrustum->getViewMatrix(eye, viewMatrix4);
MathConverterGl::convertMatrix4ToMatrixGl(viewMatrix4, viewMatrixGl);

glLoadMatrixf(viewMatrixGl);
glMultMatrixf(monoModelViewGl);
```

z space

Enabling Stereo

Adjusting The Left and Right Cameras

```
GLfloat monoModelViewGl[16];
GLfloat viewMatrixGl[16];

// Copy the mono model-view matrix.
glMatrixMode(GL_MODELVIEW);
glGetFloatv(GL_MODELVIEW_MATRIX, monoModelViewGl);

// Get the stereo frustum.
StereoFrustum* stereoFrustum = m_stereoViewport->getStereoFrustum();

// Modify the model-view matrix for eye.
// Note: The view matrix contains eye offset plus any rotation required for off-axis projection.
Matrix4 viewMatrix4;
stereoFrustum->getViewMatrix(eye, viewMatrix4);
MathConverterGl::convertMatrix4ToMatrixGl(viewMatrix4, viewMatrixGl);

glLoadMatrixf(viewMatrixGl);
glMultMatrixf(monoModelViewGl);
```

z space

Enabling Stereo

Setting The Off-Axis Projections

```
GLfloat projectionMatrixGl[16];

// Set the projection matrix for eye.
Matrix4 projectionMatrix4;
stereoFrustum->getProjectionMatrix(eye, projectionMatrix4);
MathConverterGl::convertMatrix4ToMatrixGl(projectionMatrix4,projectionMatrixGl);

glMatrixMode(GL_PROJECTION);
glLoadMatrixf(projectionMatrixGl);
```

Alternate Approach

```
StereoFrustum::Bounds bounds;

// Get the frustum bounds
bounds = stereoFrustum->getBounds(eye);

glMatrixMode(GL_PROJECTION);
glFrustum(bounds.left, bounds.right, bounds.bottom, bounds.top, bounds.nearClip, bounds.farClip);
```

Enabling Stereo

Setting The Off-Axis Projections

```
GLfloat projectionMatrixGl[16];

// Set the projection matrix for eye.
Matrix4 projectionMatrix4;
stereoFrustum->getProjectionMatrix(eye, projectionMatrix4);
MathConverterGl::convertMatrix4ToMatrixGl(projectionMatrix4,projectionMatrixGl);

glMatrixMode(GL_PROJECTION);
glLoadMatrixf(projectionMatrixGl);
```

Alternate Approach

```
StereoFrustum::Bounds bounds;

// Get the frustum bounds
bounds = stereoFrustum->getBounds(eye);

glMatrixMode(GL_PROJECTION);
glFrustum(bounds.left, bounds.right, bounds.bottom, bounds.top, bounds.nearClip, bounds.farClip);
```

Application Steps

- Stereo Enabled Rendering
- Use zSpace Stereo Data
- Enable Head Tracking
- Read Stylus
- Direct Interaction

Head Tracking

Initialize Tracking

```
// Initialize the tracking system – this is done once
m_trackerSystem = new zspace::tracker::TrackerSystem();
```

Passing Along The Head Pose

```
// Cache the current tracker targets
m_trackerSystem->captureTargets();

// Get the head target
TrackerTarget* headTarget = m_trackerSystem->getDefaultTrackerTarget(TrackerTarget::TYPE_HEAD);

// Get the head pose
Matrix4 headPose = Matrix4::IDENTITY();
headTarget->getPose(headPose);

// Set the stereo head pose
StereoFrustum* stereoFrustum = m_stereoViewport->getStereoFrustum();
stereoFrustum->setHeadPose(headPose);
```

Application Steps

- Stereo Enabled Rendering
- Use zSpace Stereo Data
- Enable Head Tracking
- Read Stylus
- Direct Interaction

Stylus Tracking

Read Stylus and Convert to World Space

```
// Get the stylus target
TrackerTarget* primaryTarget = m_trackerSystem->getDefaultTrackerTarget(TrackerTarget::TYPE_PRIMARY);

// Get the head pose
Matrix4 primaryPose= Matrix4::IDENTITY();
primaryTarget ->getPose(primaryPose);

Matrix4 trackerSpaceToCameraSpace;
Matrix4 worldPrimaryPose;

// Get the display for this window
const zspace::common::DisplayInfo::Display* display = m_stereoWindow->getCurrentDisplay();

// Get the tracker to camera space transform
trackerSpaceToCameraSpace = DisplayInfo::getTrackerToCameraSpaceTransform(display);

// Transform the pose from tracker space to world space
worldPrimaryPose = cameraToWorldSpace * trackerSpaceToCameraSpace * primaryPose;
```

Windowed Mode

Tracking Window Size and Position

```
// Assuming that the viewport is tracking the window
m_stereoWindow->move(x, y);
m_stereoWindow->resize(width, height);
```

Getting the Stylus Offset

```
Vector3 viewportOffset = Vector3::ZERO();
const zspace::common::DisplayInfo::Display* display = m_stereoWindow->getCurrentDisplay();

// Get the display offset for the given display and dimensions
viewportOffset = DisplayInfo::getViewportOffset(display, x, y, width, height);
```

Windowed Mode

Tracking Window Size and Position

```
// Assuming that the viewport is tracking the window
m_stereoWindow->move(x, y);
m_stereoWindow->resize(width, height);
```

Getting the Stylus Offset

```
Vector3 viewportOffset = Vector3::ZERO();
const zspace::common::DisplayInfo::Display* display = m_stereoWindow->getCurrentDisplay();

// Get the display offset for the given display and dimensions
viewportOffset = DisplayInfo::getViewportOffset(display, x, y, width, height);
```

Application Steps

- Stereo Enabled Rendering
- Use zSpace Stereo Data
- Enable Head Tracking
- Read Stylus
- Direct Interaction

z space®

Direct Interaction

Interpreting a Pose

```
Vector3 posePosition = Vector3::ZERO();
Vector3 poseX = Vector3::UNIT_X();
Vector3 poseY = Vector3::UNIT_Y();
Vector3 poseZ = Vector3::UNIT_Z();

Matrix4 primaryWorldPose;

// Get the position of the pose
posePosition = primaryWorldPose.getTrans();

// Get the X, Y, and Z axis
Quaternion rotation;
primaryWorldPose.extractQuaternion(rotation);
rotation.ToAxes(poseX, poseY, poseZ);
```

Direct Interaction

Mouse Emulation

```
// Get the tracker mouse simulator
TrackerMouseSimulator* trackerMouseSimulator = m_trackerSystem->getTrackerMouseSimulator();

// Set the tracker target to use
TrackerTarget* primaryTarget = m_trackerSystem->getDefaultTrackerTarget(TrackerTarget::TYPE_PRIMARY);
trackerMouseSimulator->setTrackerTarget(primaryTarget);

// Set the mouse button mappings
trackerMouseSimulator->setButtonMapping(0, TrackerMouseSimulator::MOUSE_BUTTON_LEFT);
trackerMouseSimulator->setButtonMapping(1, TrackerMouseSimulator::MOUSE_BUTTON_RIGHT);
trackerMouseSimulator->setButtonMapping(2, TrackerMouseSimulator::MOUSE_BUTTON_MIDDLE);

// Enable it
trackerMouseSimulator->setEnabled(true);
```

Questions

Transforming computer interaction

lifelike | interactive | immersive

zSpace: Direct interaction with 3D virtual-holographic simulations in open space

- Full color, high resolution stereoscopic display
- Unique stylus for 3D interaction
- Passive, polarized tracking eyewear
- Innovative software development platform

z space®