Simple, Low-Cost Stereographics: VR for Everyone

John M. Zelle and Charles Figura

Department of Mathematics, Computer Science, and Physics
Wartburg College

SIGCSE 2004

Motivation

- Virtual Reality is a hot topic
- Significant educational potential
  - Virtual travel
  - Visualizations
  - Main Requirement: multi-viewer, stereographic display
- Barriers:
  - Cost
  - Expertise
- Goal: Make this technology cheap and easy!
  - Physics perspective: educational applications
  - CS perspective: student research/design projects

Stereographics Concepts

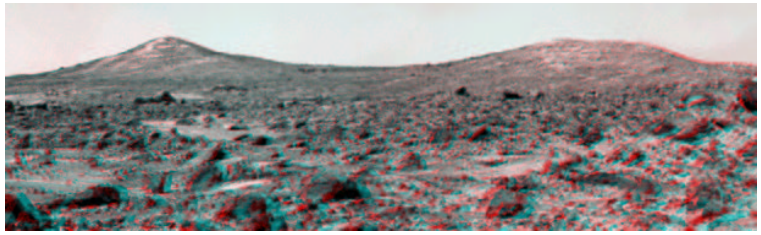
- Brain constructs 3D view of the world
- Important depth indicator: binocular vision
- Stereographic displays "fool" the brain by presenting suitable left and right eye images
- For true 3D effect must solve two problems:
  - Create correct left/right images
  - Present each eye with appropriate image

Stereographic Display Techniques

- Head-mounted display
  - Small LCD screen in front of each eye
  - Problems: Single viewer, expensive, fragile
- Active stereo
  - Stereo graphics card (quad buffered) and Shutter glasses
  - Problems: Expensive and fragile
- Passive stereo
  - Two images super-imposed
  - Inexpensive filter glasses to separate
  - Common approaches: Anaglyph (red/blue), Polarized

Anaglyphs

- Present left eye image in red, right eye in blue



- Advantages:
  - Simple display technology
  - cheap
  - multi-viewer
- Disadvantages: Black and white, eye-strain

Keystone Distortion

- Aligning centers of projection leads to keystoning
  - Side-by-side: horizontal keystoning (one side taller)
  - Stacked: vertical keystoning (top wider than bottom)
- Hardware options:
  - Lens shifting (high-end projectors only)
  - Digital keystoning correction
- Software options:
  - Image trimming
  - Software keystone correction
- Our solution: Ignore it

Introducing SVEN

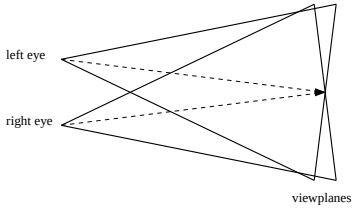
- SVEN: Stereoscopic Visualization ENvironment
- Equipment:
  - Computer
  - (2) Projectors LCD/DLP
  - (2) Polarizing Filters $30 - $200
  - Dual-head graphics card $80 - $500
  - Metallic screen $10 - $400
  - Polarized Glasses ~$0.40 per pair
- Total Cost: $200 - $1000

Stereo Applications Overview

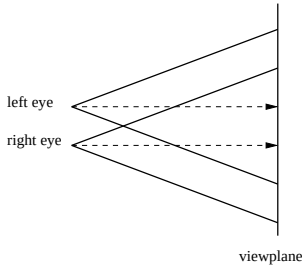
- Our "platform" has strengths and weaknesses
  - weakness: Most existing stereo-enabled programs will not work
  - strength: Most existing stereo-enabled programs will not work
- Simplest approach: paired stereo images
  - Many available on WWW (e.g. NASA)
  - Existing programs for generating images from models
- Passive stereo applications
  - Roll your own (e.g., OpenGL)
  - Adapt existing applications
  - Use an adapted framework (VPython)

Generating Stereo Pairs

- The wrong way



- The better way



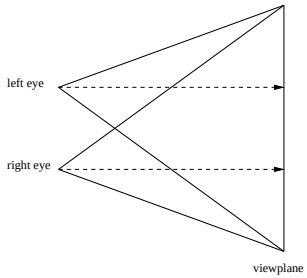
OpenGL Setting the Projection

```
def drawEye(self, sgn, r, aspect):
    # sgn is -1 for left eye, 1 for right eye
    # r is eye_separation/2.0 maginitude vector
    #        pointing to camera's right
    # aspect is the aspect ratio

    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    eyeOff = (sgn * (self.eyesep / 2.0)
              * (self.near/self.focallength))
    top = self.near * math.tan(self.fov/2.0))
    right = aspect*top
    glFrustum(-right-eyeOff, right-eyeOff,
              -top, top,
              self.near, self.far)
```

Using OpenGL

- OpenGL allows off-axis (asymmetric) view frustum
- The "best" way



- Using glFrustum

```
glFrustum(left, right, bottom, top, near, far)
```

OpenGL (cont'd) Setting the View

```
# set the lookat point (view)
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
vp = self.vp + (sgn * r)
lookat = vp + self.vdir()
up = self.vu
gluLookAt(vp[X], vp[Y], vp[Z],
          lookat[X], lookat[Y], lookat[Z],
          up[X], up[Y], up[Z])
self.display() # Execute drawing primitives
```

Example Applications: dimg and panner

- dimg displays paired images sized to the display
  - input: left-right images, anaglyph, "pickle" file
  - modes: passive stereo, anaglyph, interlaced
- panner is similar, but does panning and zooming
- Both written in Python using Python Imaging Library

Stereo Without Graphics: VPython

- VPython: 3D Programming for Ordinary Mortals
  - C++ extension module for Python (uses OpenGL)
  - Provides a set of primitives for 3D Modeling (Box, Sphere, Cone, Line, etc)
  - Provides vector arithmetic
  - Automatically manages view in a separate thread
- VPython stereo mode (now in standard distribution)
  - scene.stereo = <'passive', 'active', 'redblue', ...>
  - scene.stereodepth = <scaled focal length 0-2>
  - scene.fullscreen = True

Example Application: PyVRML3D

- Student project to render VRML in 3D
  - input: subset of VRML 97
  - modes: passive, anaglyph, interlaced
  - Allows scene navigation
  - Written in Python using PyOpenGL

VPython Example: Bounce

```
from visual import *

floor = box(length=4, height=0.5,
            width=4, color=color.blue)

ball = sphere(pos=(0,4,0), color=color.red)
ball.velocity = vector(0,-1,0)

scene.autoscale=0
dt = 0.01
while True:
    rate(100)
    ball.pos = ball.pos + ball.velocity*dt
    if ball.y < 1:
        ball.velocity.y = -ball.velocity.y
    else:
        ball.velocity.y = ball.velocity.y - 9.8*dt
```

VPython Example: Stereo Bounce

```
scene.stereo = 'passive'
scene.stereodepth = 1.5

floor = box(length=4, height=0.5,
            width=4, color=color.blue)
ball = sphere(pos=(0,4,0), color=color.red)
ball.velocity = vector(0,-1,0)
scene.autoscale=0
dt = 0.01
while True:
    rate(100)
    ball.pos = ball.pos + ball.velocity*dt
    if ball.y < 1:
        ball.velocity.y = -ball.velocity.y
    else:
        ball.velocity.y = ball.velocity.y - 9.8*dt
```

Conclusions

- 3D Visualization can be done cheaply and easily
- Students love it
- You should give it a try!
- Find our materials at http://mcsp.wartburg.edu/SVEN
- Acknowledgements
  - Student researchers: L. Blake, A. Goerdt, J. Oltrogge, A. Hammond, D. Lindner
  - Maytag Corporation Funding: Innovation awards for undergraduate research program.

VPython Example: Stereo Bounce

```
scene.stereo = 'redblue'
scene.stereodepth = 1.5

floor = box(length=4, height=0.5,
            width=4, color=color.blue)
ball = sphere(pos=(0,4,0), color=color.red)
ball.velocity = vector(0,-1,0)
scene.autoscale=0
dt = 0.01
while True:
    rate(100)
    ball.pos = ball.pos + ball.velocity*dt
    if ball.y < 1:
        ball.velocity.y = -ball.velocity.y
    else:
        ball.velocity.y = ball.velocity.y - 9.8*dt
```