http://vision.eng.shu.ac.uk/jan/shrug15.pdf

Play Squash with Ruby, OpenGL, and a Wiimote

Play Squash with Ruby, OpenGL and a Wiimote

Jan Wedekind

Mon Feb 14 19:00:00 BST 2011

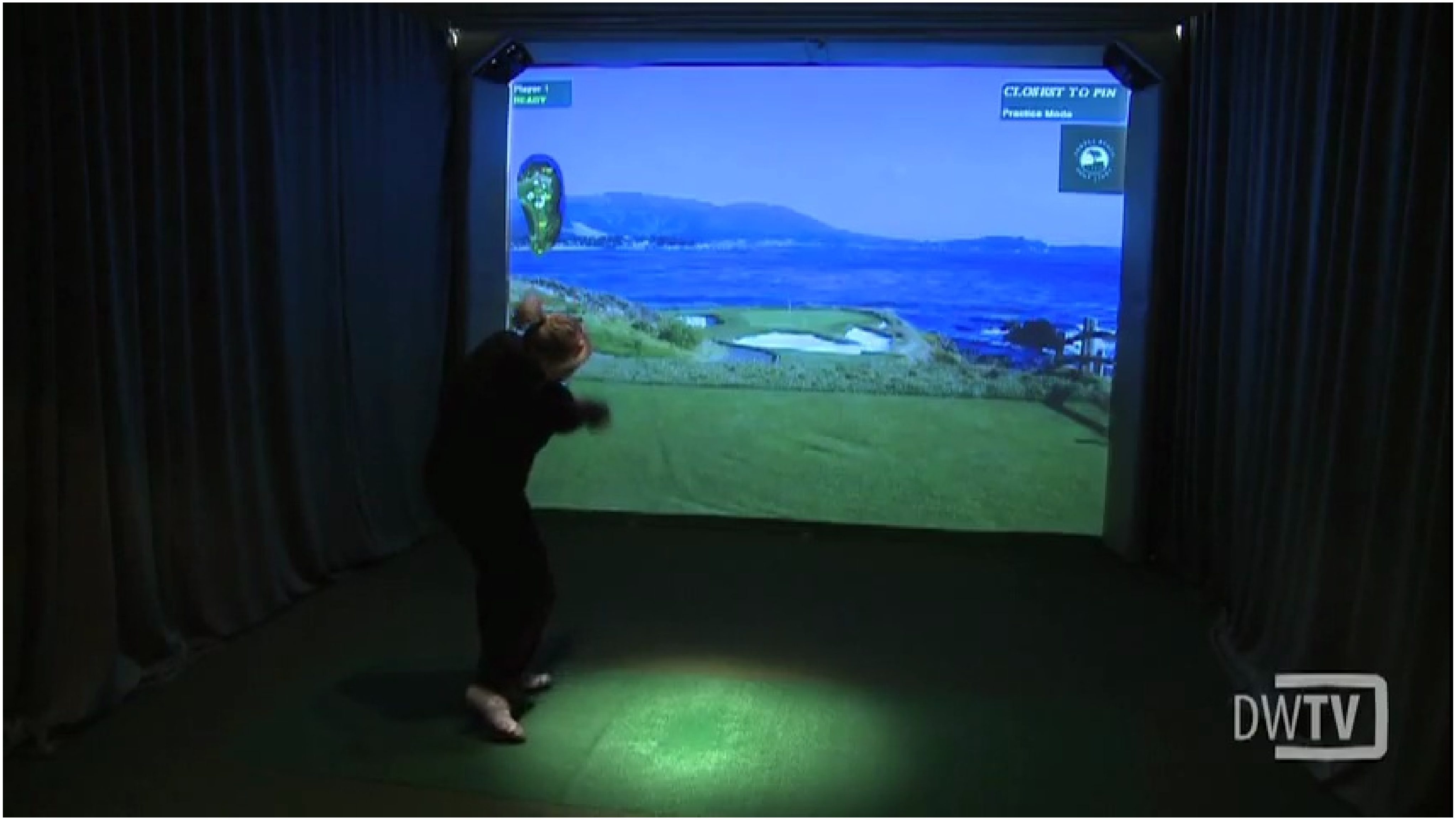
Golf Simulator

Outland

Movie scene from “Outland”

Golf Simulator

High Definition Golf^{TM}



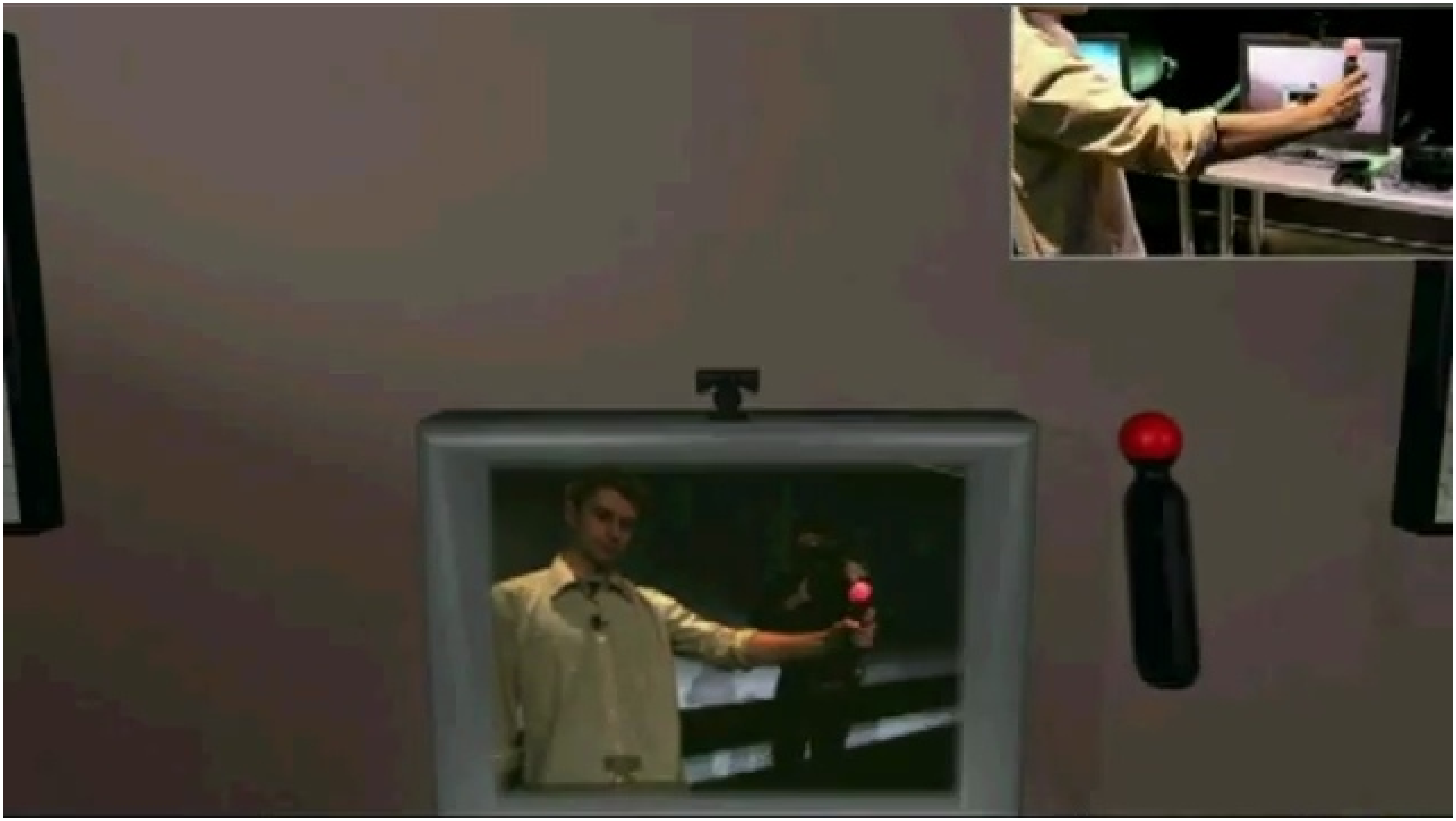
Golf Simulator (using high-speed cameras?)

Sony PS3 Move

Bluetooth device

- 8 buttons (△, ○, ×, □, Select, Home, Move)
- Analog trigger
- Sphere illuminated by RGB LED for external motion tracking
- 3-axis accelerometer
- 3-axis gyroscope
- magnetometer

PS3 Move E3 Tech Demo



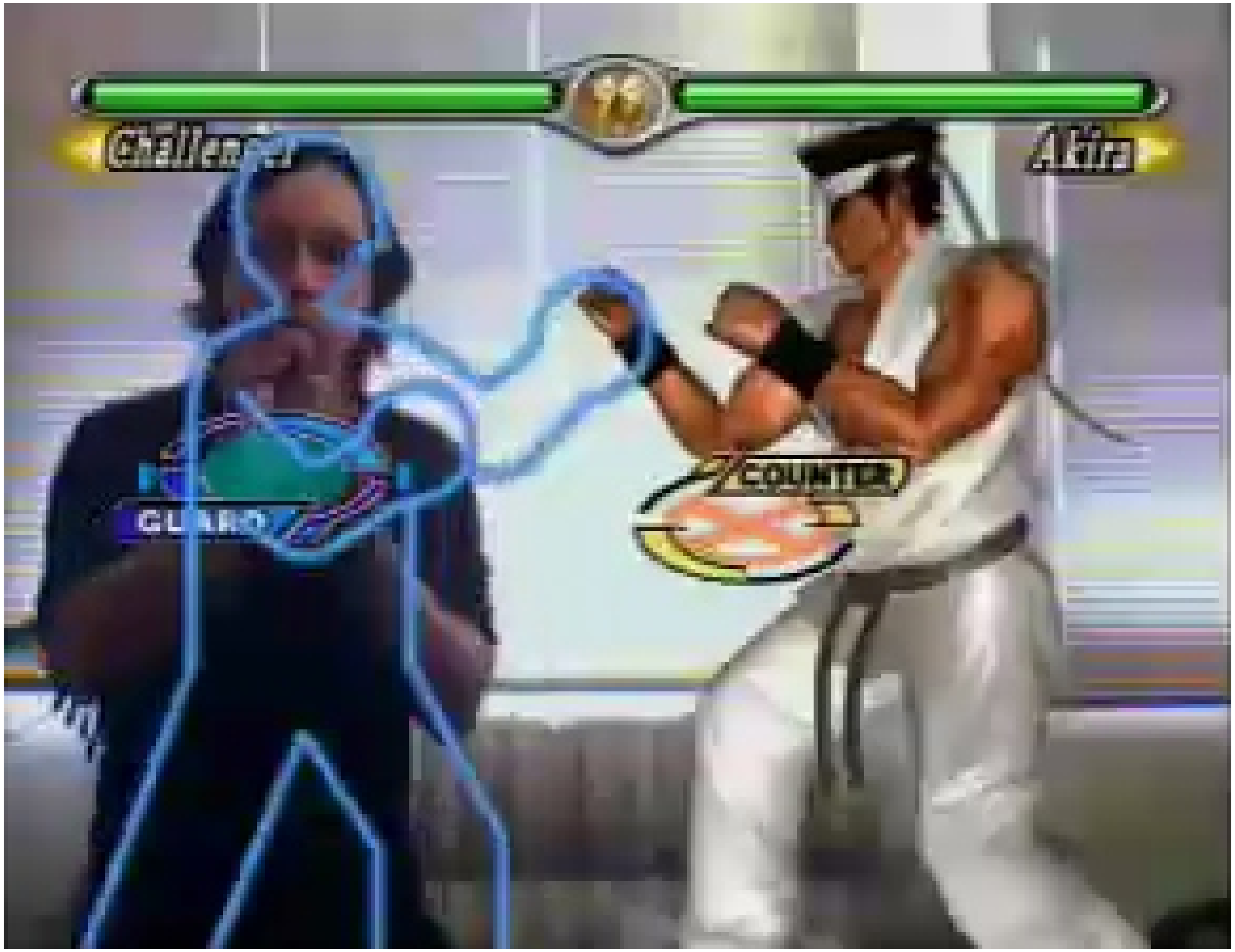
Also see

http://howtohackps3.com/playstation-move-new-tech-demo-video-tour/

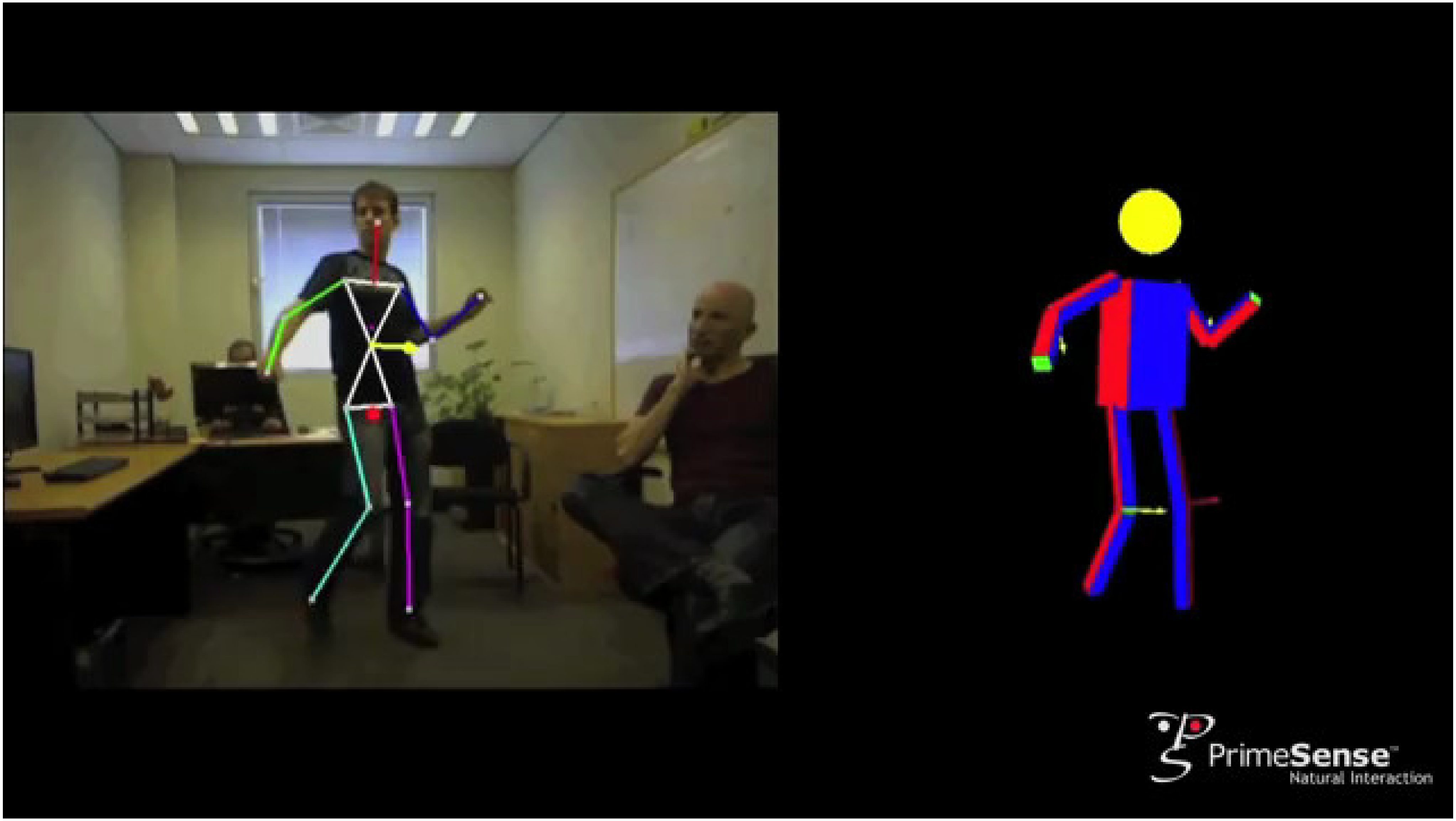
Sony EyeToy

Webcam

Sega Virtua Fighter for EyeToy



Microsoft Kinect



Also see http://www.primesense.com/

Camspace

Webcam instead of Game Controller

The Nintendo Wii Remote (Wiimote)

Bluetooth device

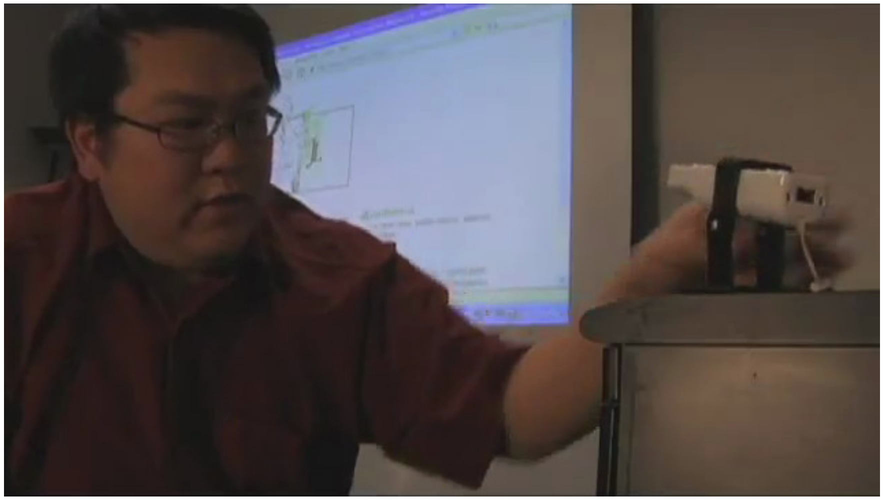
- 11 buttons (1, 2, A, B, +, -, Home, ⇐, ⇒, ⇑, ⇓)
- 4 LEDs
- rumble motor
- 3-axis accelerometer
- IR camera and chip for detecting 5 brightest dots
- speaker
- expansion port

Wii E3 Tech Demo

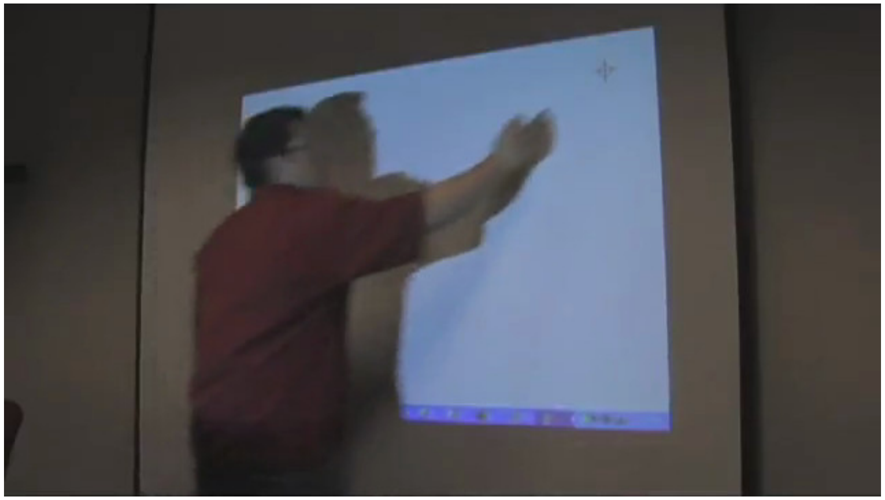
Wiimote Whiteboard by Johnny Chung Lee

IR pen

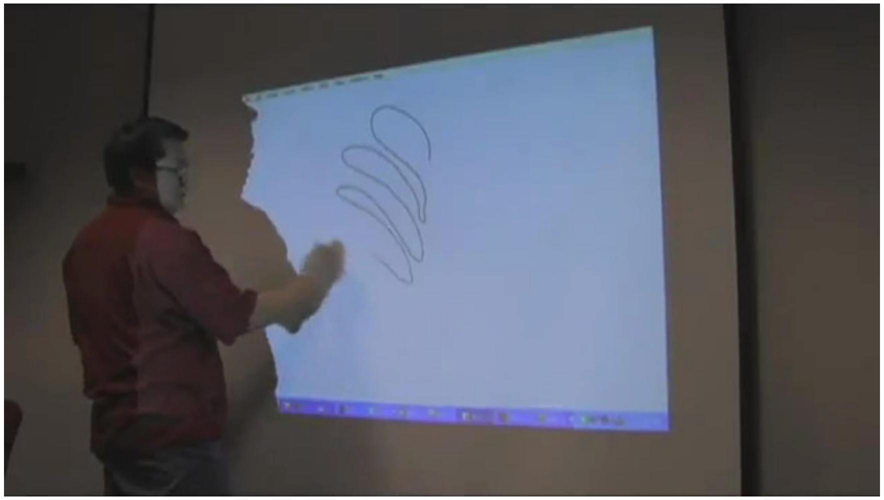
Wiimote stand



4 point calibration



⇒ Wiimote Whiteboard



See http://johnnylee.net/projects/wii/

Cwiid library

```
#include <cwiid.h>
#include <stdio.h>
int main(void) // gcc -o wii wii.c -lcwiid
{
  cwiid_wiimote_t *wiimote;
  struct cwiid_state state;
  bdaddr_t any = { { 0, 0, 0, 0, 0, 0 } };
  wiimote = cwiid_open( &any, 0 );
  cwiid_set_rpt_mode( wiimote, CWIID_RPT_BTN | CWIID_RPT_ACC );
  do {
    cwiid_get_state( wiimote, &state );
    printf( "{ %3d, %3d, %3d }\n",
            state.acc[0], state.acc[1], state.acc[2] );
    usleep( 10000 );
  } while ( ( state.buttons & CWIID_BTN_A ) == 0 );
  cwiid_close( wiimote );
  return 0;
}
```

See http://abstrakraft.org/cwiid/ for Cwiid library

See http://wiibrew.org/wiki/Wiimote/Library for other libraries

Cwiid Ruby Gem

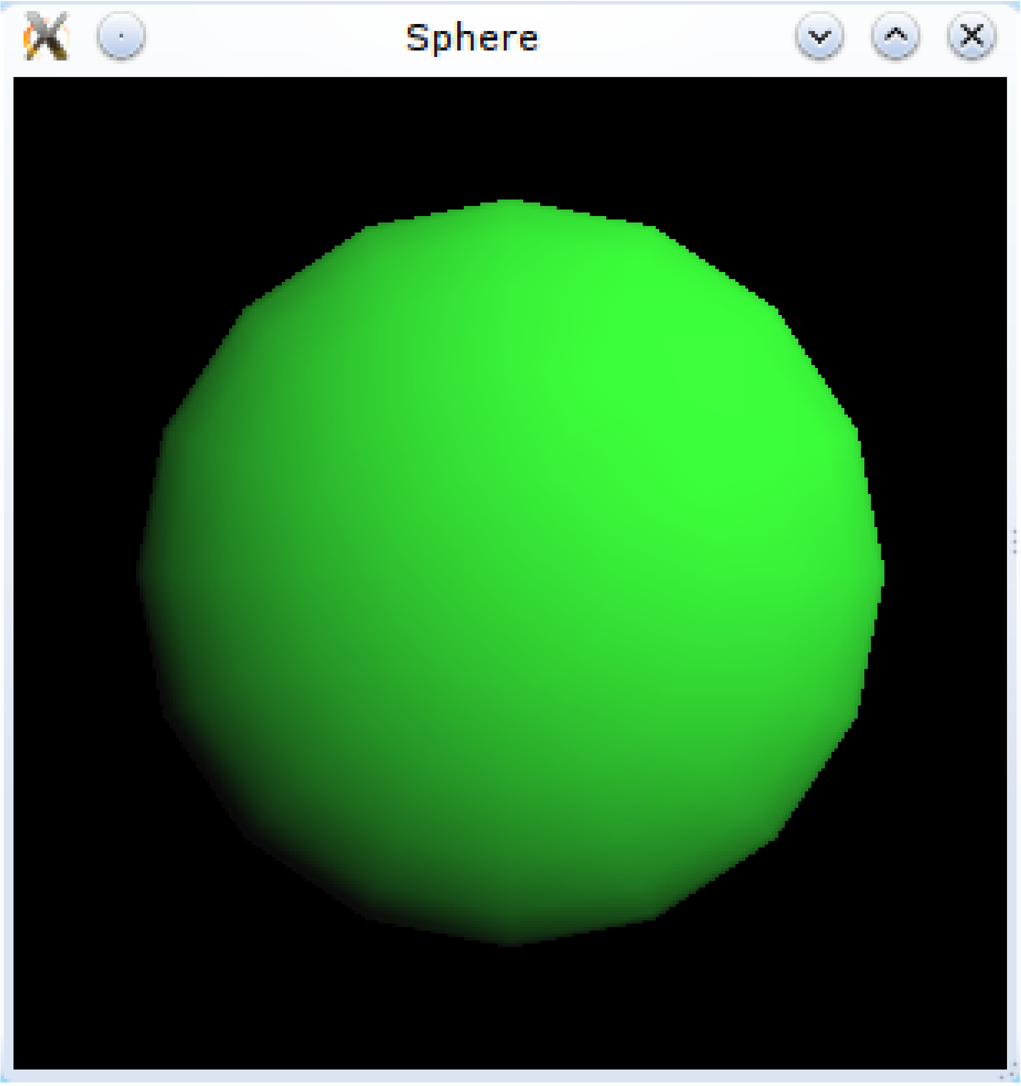
```ruby
require 'cwiid' # ruby -rrubygems wii.rb
wiimote = WiiMote.new
wiimote.rpt_mode = WiiMote::RPT_BTN | WiiMote::RPT_ACC
begin
  wiimote.get_state
  puts "[ %3d, %3d, %3d ]" % wiimote.acc
end until ( wiimote.buttons & WiiMote::BTN_A ) != 0
wiimote.close
```

See https://github.com/wedesoft/cwiid/ for Cwiid Ruby Gem

OpenGL with Ruby

```ruby
require 'opengl' # http://ruby-opengl.rubyforge.org/
display = proc do
  GL.Clear GL::COLOR_BUFFER_BIT | GL::DEPTH_BUFFER_BIT
  GL.Material GL::FRONT, GL::DIFFUSE, [ 0.2, 1.0, 0.2, 1.0 ]
  GLUT.SolidSphere 1.0, 16, 16
  GLUT.SwapBuffers
end
reshape = proc do |w, h|
  GL.Viewport 0, 0, w, h
  GL.MatrixMode GL::PROJECTION
  GL.LoadIdentity
  GLU.Perspective 60.0, w.to_f / h, 1.0, 20.0
  GL.MatrixMode GL::MODELVIEW
  GL.LoadIdentity
  GL.Translate 0.0, 0.0, -2.5
end
GLUT.Init
GLUT.InitDisplayMode GLUT::DOUBLE | GLUT::RGB | GLUT::DEPTH
GLUT.CreateWindow 'Sphere'
GL.Light GL::LIGHT0, GL::POSITION, [ 0.5, 0.5, 1.0, 0.0 ]
GL.Enable GL::LIGHTING
GL.Enable GL::LIGHT0
GL.Enable GL::DEPTH_TEST
GLUT.DisplayFunc display
GLUT.ReshapeFunc reshape
GLUT.MainLoop
```

OpenGL with Ruby



Also see “Red Book”: http://vision.eng.shu.ac.uk/jan/redbook.pdf

Ruby OpenGL Gem comes with code examples

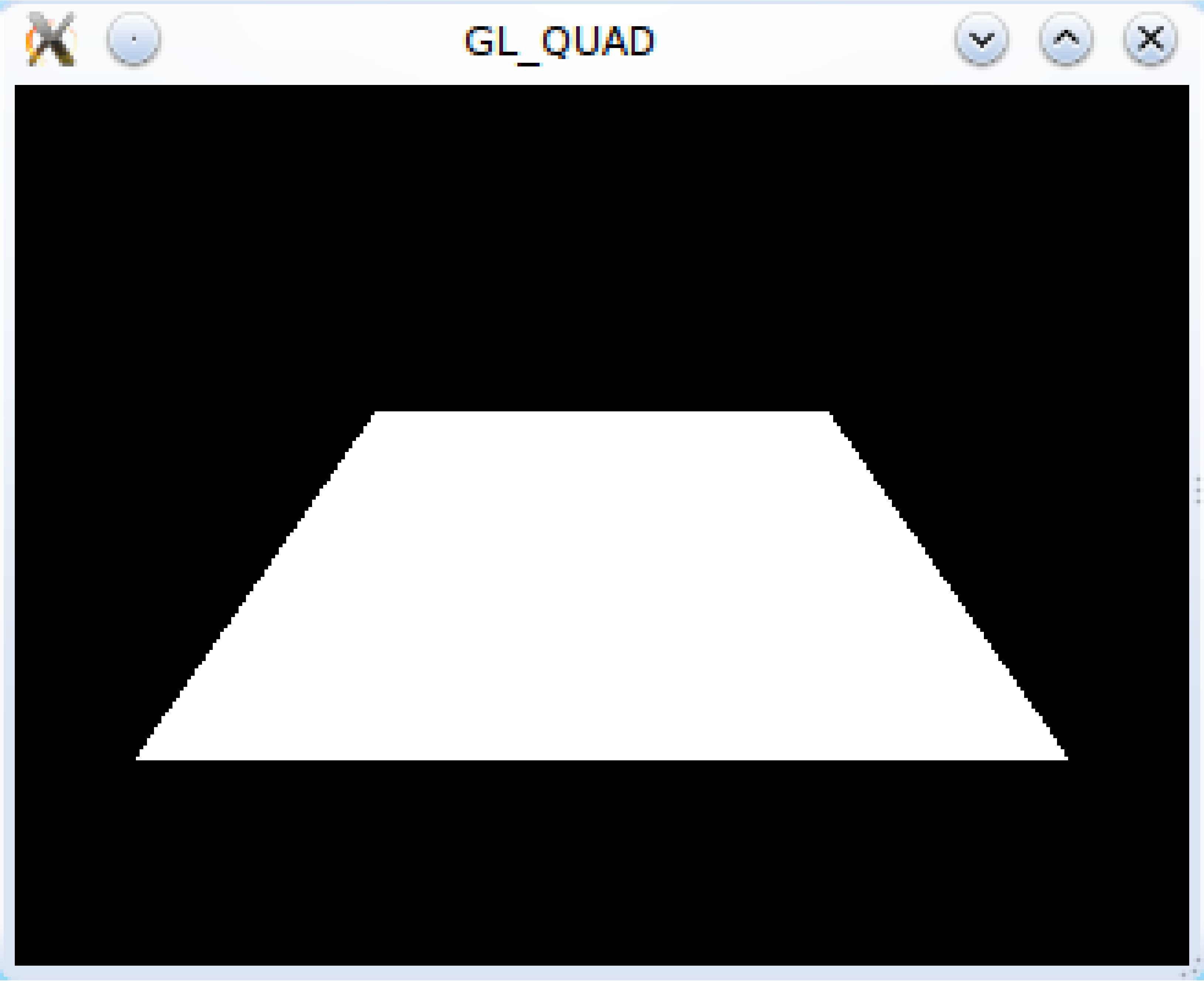
GLUT Framework

```
require 'opengl'
$list = nil
def init
  $list = GL.GenLists 1
  GL.NewList $list, GL_COMPILE
  GL.EndList
end
display = proc do
  GL.Clear GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT
  GL.CallList $list
  GLUT.SwapBuffers
end
reshape = proc { |w, h| }
keyboard = proc { |key, x, y| }
animate = proc { GLUT.PostRedisplay }
GLUT.Init
GLUT.InitDisplayMode GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH
GLUT.InitWindowSize 640, 480
GLUT.CreateWindow 'Test'
init
GLUT.DisplayFunc display
GLUT.ReshapeFunc reshape
GLUT.KeyboardFunc keyboard
GLUT.IdleFunc animate
GLUT.MainLoop
```

Quadruple of Vertices

```
# ...
def init
  $list = GL.GenLists 1
  GL.NewList $list, GL_COMPILE
  GL.Begin GL::QUADS
  GL.Normal  0.0, 1.0, 0.0
  GL.Vertex -1.0,-1.0, 0.0
  GL.Vertex  1.0,-1.0, 0.0
  GL.Vertex  1.0, 1.0, 0.0
  GL.Vertex -1.0, 1.0, 0.0
  GL.End
  GL.EndList
end
display = proc do
  GL.Clear GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT
  GL.PushMatrix
  GL.Rotate -60.0, 1.0, 0.0, 0.0
  GL.CallList $list
  GL.PopMatrix
  GLUT.SwapBuffers
end
# ...
```

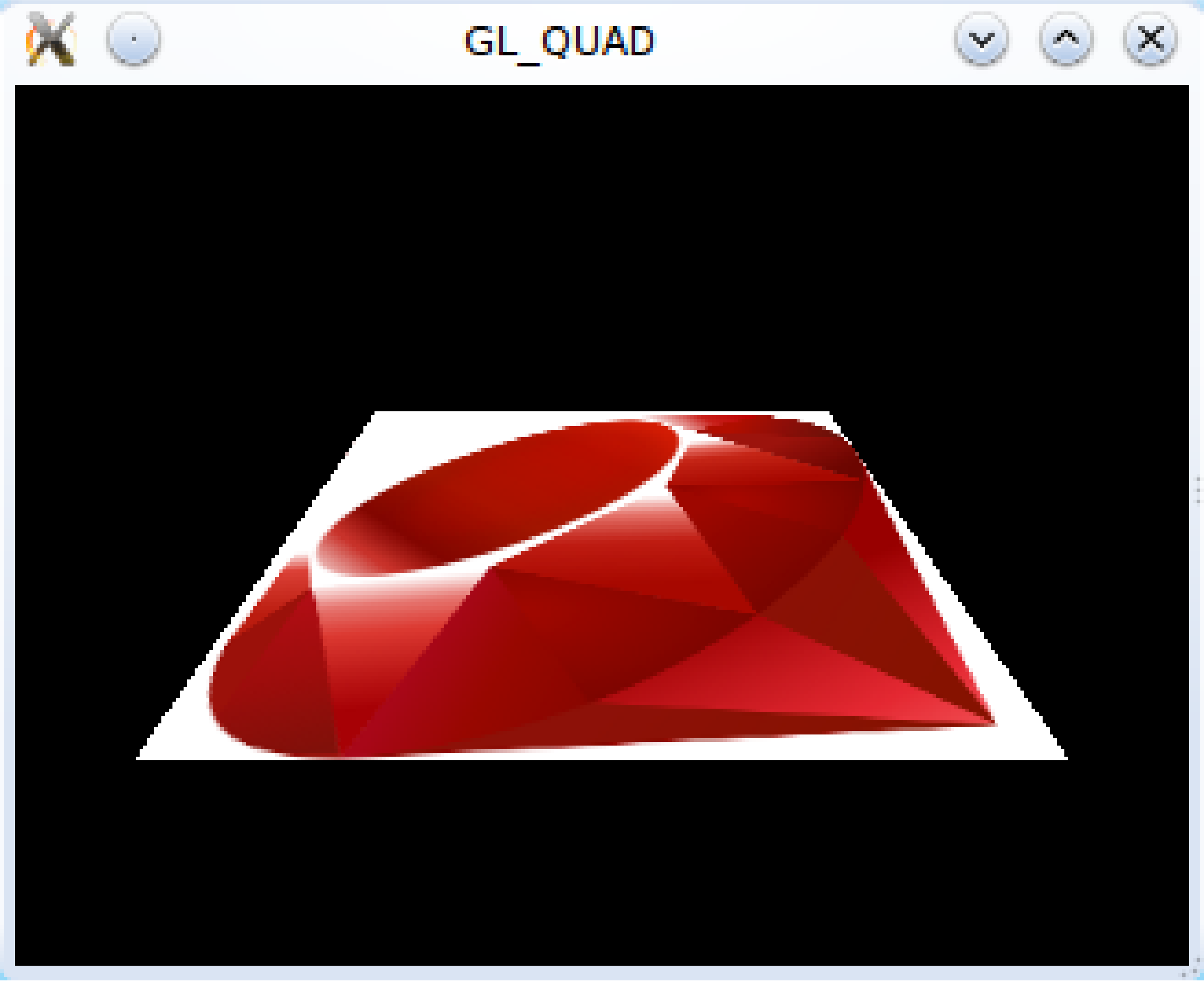
Quadruple of Vertices



Texture Mapping

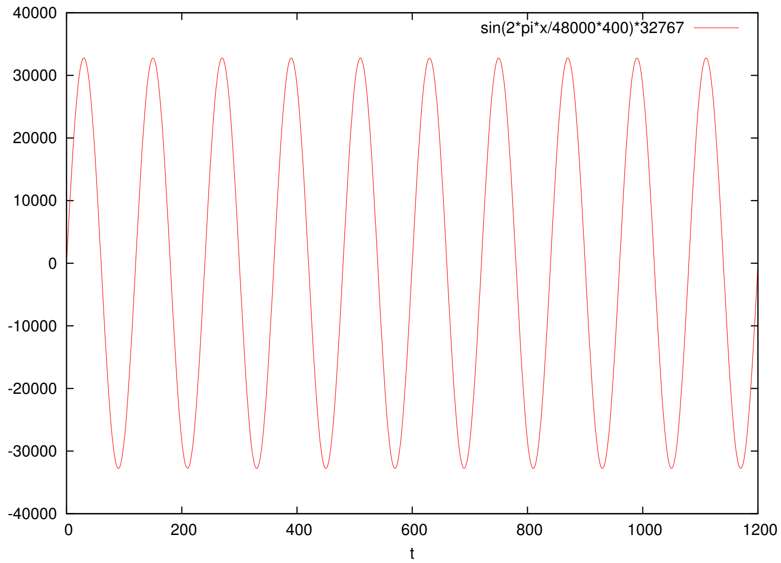
```
# ...
def init
  $tex = GL.GenTextures 1
  image = Magick::Image.read( 'ruby.png' ).first
  GL.BindTexture GL::TEXTURE_2D, $tex[0]
  GL.TexParameter GL::TEXTURE_2D, GL::TEXTURE_MIN_FILTER, GL::NEAREST
  GL.TexImage2D GL::TEXTURE_2D, 0, GL::RGB, image.columns, image.rows, 0,
                GL::RGB, GL::UNSIGNED_BYTE,
                image.export_pixels_to_str( 0, 0, image.columns, image.rows,
                                            'RGB', Magick::CharPixel )
  $list = GL.GenLists 1
  GL.NewList $list, GL_COMPILE
  GL.Enable GL::TEXTURE_2D
  GL.Material GL::FRONT, GL::AMBIENT, [ 1.0, 1.0, 1.0, 1.0 ]
  GL.BindTexture GL::TEXTURE_2D, $tex[0]
  GL.Begin GL::QUADS
  GL.Normal  0.0, 1.0, 0.0
  GL.TexCoord 0.0, 1.0; GL.Vertex -1.0,-1.0, 0.0
  GL.TexCoord 1.0, 1.0; GL.Vertex  1.0,-1.0, 0.0
  GL.TexCoord 1.0, 0.0; GL.Vertex  1.0, 1.0, 0.0
  GL.TexCoord 0.0, 0.0; GL.Vertex -1.0, 1.0, 0.0
  GL.End
  GL.Disable GL::TEXTURE_2D
  GL.EndList
end
# ...
```

Texture Mapping



ALSA

```
require 'hornetseye_alsa'
include Hornetseye
RATE = 48000
CHANNELS = 2
LEN = RATE / 400
output = AlsaOutput.new 'default:0', RATE, CHANNELS
sine = lazy( CHANNELS, LEN ) do |i,j|
  Math.sin( 2 * Math::PI * j / LEN ) * 32767
end.to_sint
( 3 * RATE / LEN ).times { output.write sine }
output.drain
```



WAV Files

```
require 'hornetseye_alsa'
include Hornetseye
RATE = 11025
CHANNELS = 1
def wav( file_name )
  str = File.new( file_name, 'rb' ).read[ 44 .. -1 ]
  malloc = Malloc.new str.size
  malloc.write str
  MultiArray( SINT, CHANNELS,
              malloc.size / ( 2 * CHANNELS ) ).new malloc
end
output = AlsaOutput.new 'default:0', RATE, CHANNELS
output.write wav( 'forest.wav' )
output.drain
```

Squash

Karim Darwish vs Gregory Gaultier

Bouncing Ball

```
# ...
G = 9.81; F = 0.8; R = 0.3
$y = 2;   $vy = 0; $t = Time.new.to_f
display = proc do
  GL.Clear GL::COLOR_BUFFER_BIT | GL::DEPTH_BUFFER_BIT
  GL.PushMatrix
  GL.Material GL::FRONT, GL::DIFFUSE, [ 0.2, 1.0, 0.2, 1.0 ]
  GL.Translate 0.0, $y, 0.0
  GLUT.SolidSphere R, 16, 16
  GL.PopMatrix
  GLUT.SwapBuffers
end
animate = proc do
  dt = Time.new.to_f - $t
  $t += dt
  $vy -= G * dt
  $y += ( $vy - 0.5 * G * dt ) * dt
  if $y < R
    $y = 2 * R - $y
    $vy = -F * $vy
  end
  GLUT.PostRedisplay
end
# ...
```

$$\Delta t = t_{i+1} - t_i$$

$$\vec{x}_{i+1} = \vec{x}_i + \vec{v}_i\,\Delta t + \tfrac{1}{2}\,\vec{a}_i\,\Delta t^2$$

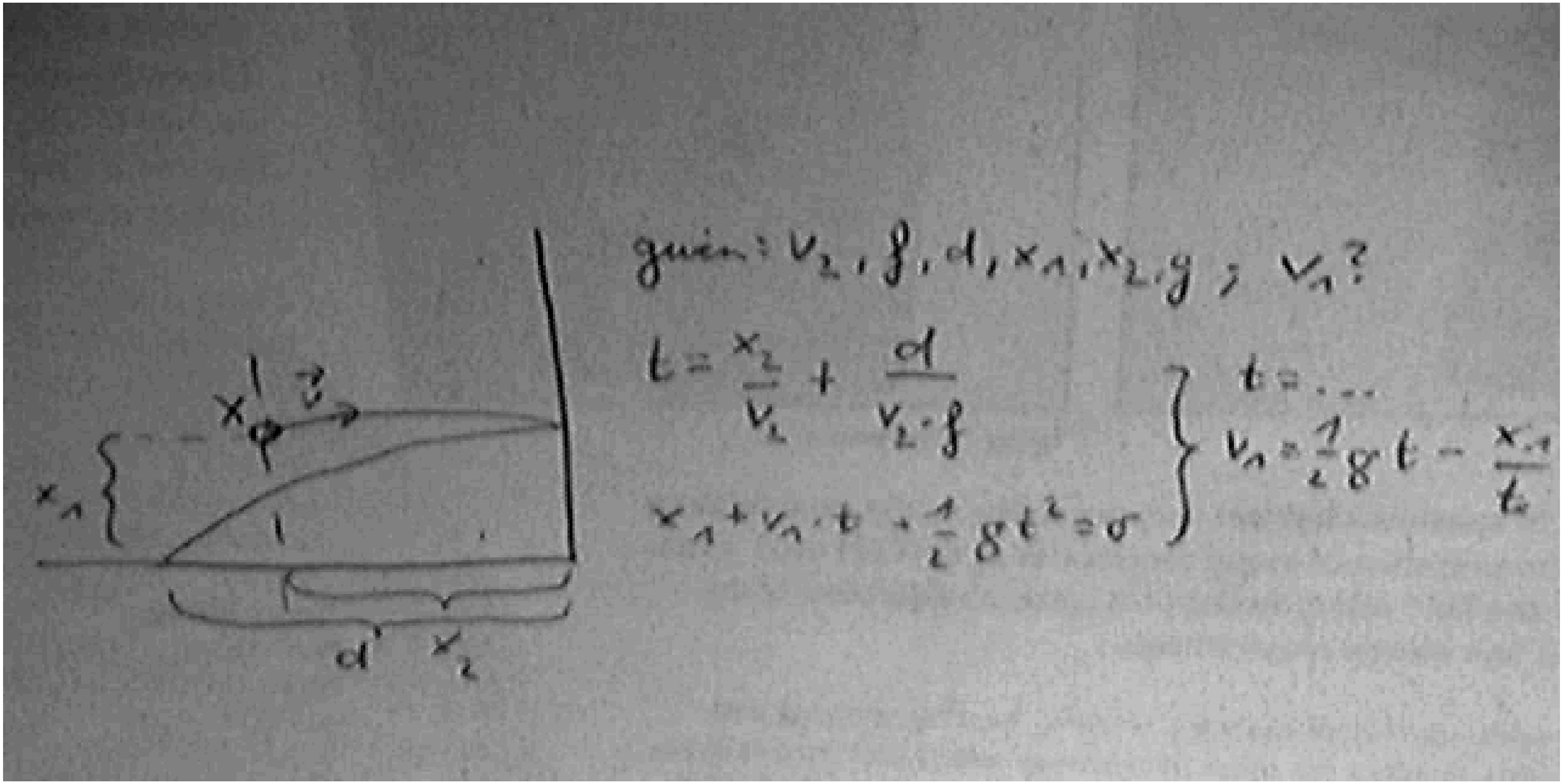
$$\vec{v}_{i+1} = \vec{v}_i + \vec{a}_i\,\Delta t$$

$$\vec{a}_i = \begin{pmatrix} 0 \\ -g \\ 0 \end{pmatrix}$$

Strike: Strength and Time

```
require 'hornetseye_alsa'
require 'cwiid'
include Hornetseye
RISING = 20.0; FALLING = 0.0; DELAY = 0.5
s = File.new( 'racket.wav', 'rb' ).read[ 44 .. -1 ]
m = Malloc.new s.size; m.write s
racket = MultiArray( SINT, 2, m.size / 4 ).new m
wiimote = WiiMote.new
output = AlsaOutput.new
wiimote.rpt_mode = WiiMote::RPT_BTN | WiiMote::RPT_ACC
sign = nil; strength = 0.0; delay = Time.new.to_f
begin
  wiimote.get_state
  acc = wiimote.acc.collect { |x| ( x - 120.0 ) / 2.5 }
  if acc[2].abs >= RISING and Time.new.to_f >= delay
    sign = acc[2] > 0 ? +1 : -1 unless sign
    strength = [ acc[2].abs, strength ].max
  elsif sign
    if acc[2] * sign <= FALLING
      output.write( ( racket * ( strength / 120.0 ) ).to_sint )
      puts "strength = #{strength}"
      sign = nil; strength = 0.0; delay = Time.new.to_f + DELAY
    end
  end
end until wiimote.buttons == WiiMote::BTN_B
```

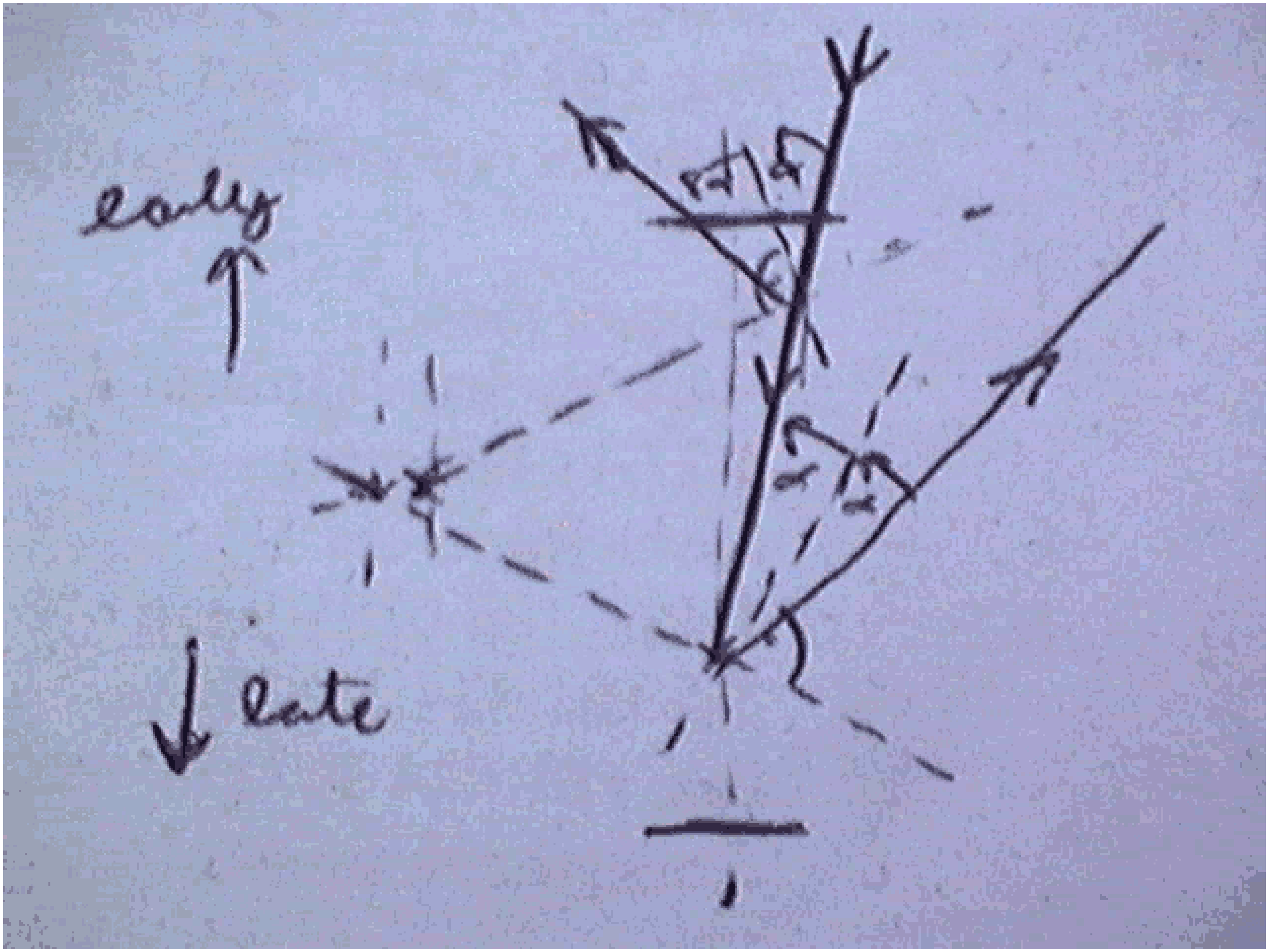
Constraints



given: v_2, f, d, x_1, x_2, g ; v_1?

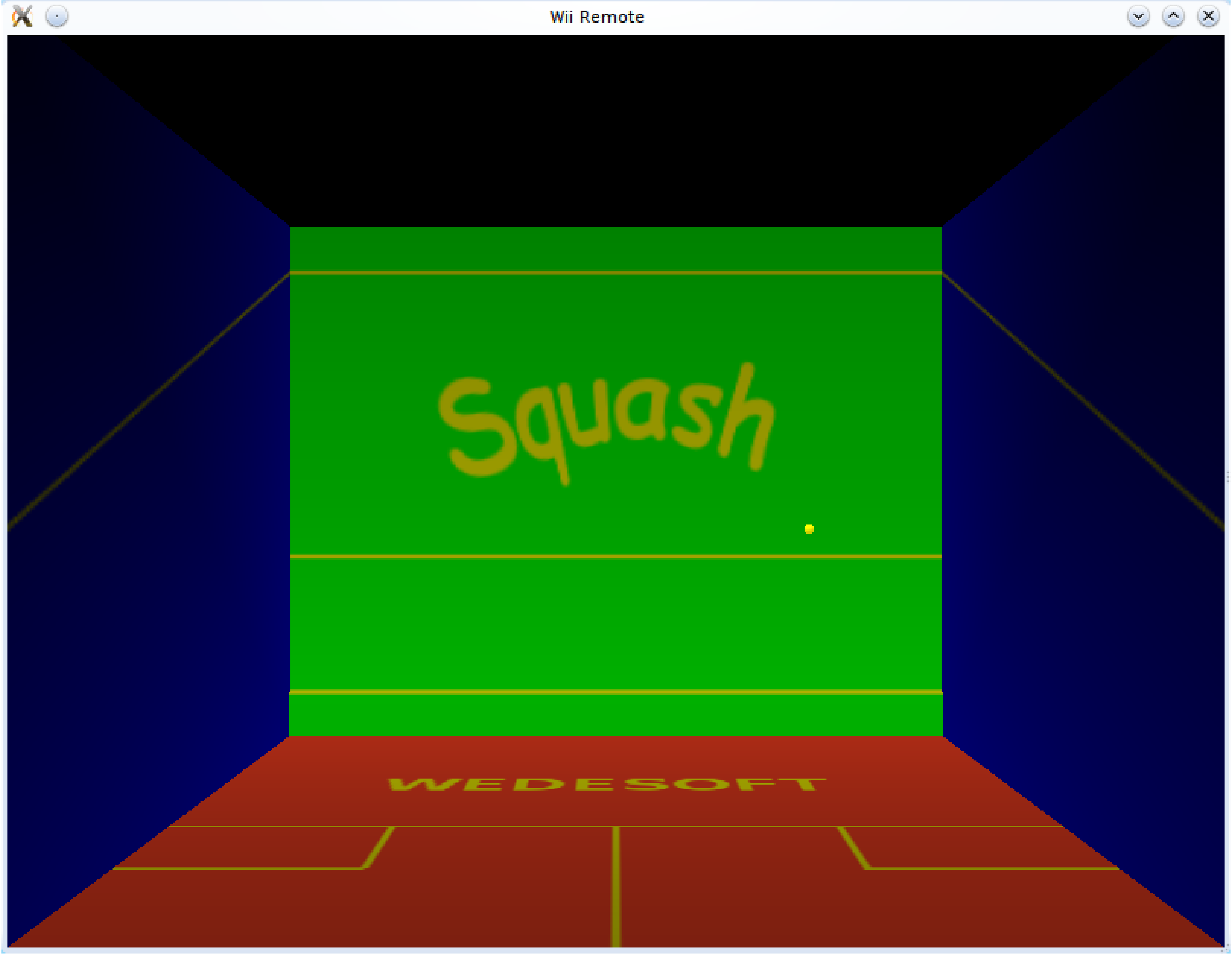
$$\left.\begin{array}{l} t = \frac{x_2}{v_2} + \frac{d}{v_2 \cdot f} \\ x_1 + v_1 \cdot t - \frac{1}{2} g t^2 = 0 \end{array}\right\} \begin{array}{l} t = \ldots \\ v_1 = \frac{1}{2} g t - \frac{x_1}{t} \end{array}$$

Constraints



http://vision.eng.shu.ac.uk/jan/shrug15.pdf

Demo



TODO

- OpenGL shadows
- scoring system
- player visualisation
- multiplayer
- better physics (Runge-Kutta integration)
- …