# Department of Computer Science
## Undergraduate Events

<table>
<thead>
<tr>
<th>Events this week</th>
<th>RIM Info Session</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Resume Editing Drop-In Session</strong></td>
<td><strong>Date:</strong> Thurs., Feb 4</td>
</tr>
<tr>
<td><strong>Date:</strong> Mon., Feb 1</td>
<td><strong>Time:</strong> 5:30 – 7 pm</td>
</tr>
<tr>
<td><strong>Time:</strong> 11 am – 2 pm</td>
<td><strong>Location:</strong> DMP 110</td>
</tr>
<tr>
<td><strong>Location:</strong> Rm 255, ICICS/CS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>EADS Info Session</strong></th>
<th><strong>Events next week</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date:</strong> Mon., Feb 1</td>
<td><strong>Finding a Summer Job or</strong></td>
</tr>
<tr>
<td><strong>Time:</strong> 3:30 – 5:30 pm</td>
<td><strong>Internship Info Session</strong></td>
</tr>
<tr>
<td><strong>Location:</strong> CEME 1202</td>
<td><strong>Date:</strong> Wed., Feb 10</td>
</tr>
<tr>
<td></td>
<td><strong>Time:</strong> 12 pm</td>
</tr>
<tr>
<td></td>
<td><strong>Location:</strong> X836</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Job Interview Practice Session</strong></th>
<th><strong>Masters of Digital Media</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>(for non-coop students)</em></td>
<td><strong>Program Info Session</strong></td>
</tr>
<tr>
<td><strong>Date:</strong> Tues., Feb 2</td>
<td><strong>Date:</strong> Thurs., Feb 11</td>
</tr>
<tr>
<td><strong>Time:</strong> 11 am – 1 pm</td>
<td><strong>Time:</strong> 12:30 – 1:30 pm</td>
</tr>
<tr>
<td><strong>Location:</strong> Rm 206, ICICS/CS</td>
<td><strong>Location:</strong> DMP 201</td>
</tr>
</tbody>
</table>
Project 1 Grading News

- don’t forget to show up 5 min before your slot
  - see news item on top of course page for signup sheet scan
- if you have not signed up or need to change your time, contact shailen AT cs.ubc.ca
  - you will lose marks if we have to hunt you down!
Review: Perspective Warp/Predistortion

• perspective viewing frustum predistorted to cube
• orthographic rendering of warped objects in cube produces same image as perspective rendering of original frustum
Review: Separate Warp and Homogenize

- warp requires only standard matrix multiply
  - distort such that orthographic projection of distorted objects shows desired perspective projection
    - w is changed
  - clip after warp, before divide
  - division by w: homogenization
Review: Perspective to NDCS Derivation

- shear
- scale
- projection-normalization

\[
\begin{bmatrix}
\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n}{t-b} & \frac{r-l}{t-b} & 0 \\
0 & 0 & \frac{-(f+n)}{f-n} & -\frac{2fn}{f-n} \\
0 & 0 & \frac{f-n}{f-n} & 0
\end{bmatrix}
\]
Review: N2D Transformation

\[
\begin{bmatrix}
x_D \\
y_D \\
z_D \\
1
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & \frac{width}{2} - \frac{1}{2} \\
0 & 1 & 0 & \frac{height}{2} - \frac{1}{2} \\
0 & 0 & 1 & \frac{depth}{2} \\
0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
width/2 & 0 & 0 & 0 \\
height/2 & 0 & 0 & 0 \\
depth/2 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x_N \\
y_N \\
z_N \\
1
\end{bmatrix} = \begin{bmatrix}
\frac{width(x_N + 1) - 1}{2} \\
\frac{height(-y_N + 1) - 1}{2} \\
\frac{2}{depth(z_N + 1)} \\
2/1
\end{bmatrix}
\]
Review: Projective Rendering Pipeline

OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

modeling transformation

W2V

viewing transformation

V2C alter w

projection transformation

C2N

perspective division

N2D

viewport transformation

O2W

glTranslatef(x,y,z)
glRotatef(a,x,y,z)

....

glVertex3f(x,y,z)

....

gluLookAt(...)

glFrustum(...)
Perspective Example

\[
\begin{bmatrix}
\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\
0 & 0 & -\frac{(f+n)}{f-n} & -\frac{2fn}{f-n} \\
0 & 0 & -1 & 0 \\
\end{bmatrix}
\]

view volume
- left = -1, right = 1
- bot = -1, top = 1
- near = 1, far = 4

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & -\frac{5}{3} & -\frac{8}{3} & 0 \\
0 & 0 & -1 & 0 & 0 \\
\end{bmatrix}
\]
tracks in VCS:
  left  x=-1, y=-1
  right x=1, y=-1

view volume
  left = -1,  right = 1
  bot = -1,  top = 1
  near = 1,  far = 4

Perspective Example
Perspective Example

\[
\begin{bmatrix}
1 \\
-1 \\
-5z_{VCS}/3 - 8/3 \\
-z_{VCS}
\end{bmatrix}
= \begin{bmatrix}
1 \\
1 \\
-5/3 \\
-1
\end{bmatrix}
\begin{bmatrix}
1 \\
-5/3 \\
-8/3 \\
z_{VCS} \\
1
\end{bmatrix}
\]

\[x_{NDCS} = -\frac{1}{z_{VCS}}\]
\[y_{NDCS} = \frac{1}{z_{VCS}}\]
\[z_{NDCS} = \frac{5}{3} + \frac{8}{3z_{VCS}}\]
OpenGL Example

```
# CCS
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
gluPerspective( 45, 1.0, 0.1, 200.0 );

# VCS
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
glTranslatef( 0.0, 0.0, -5.0 );

# WCS
glPushMatrix();
glTranslate( 4, 4, 0 );

# OCS1
glutSolidTeapot(1);

# OCS2
glutSolidTeapot(1);
```

Transformations that are applied to object first are specified last.
Viewing: More Camera Motion
Fly "Through The Lens": Roll/Pitch/Yaw
Viewing: Incremental Relative Motion

• how to move relative to current camera coordinate system?
  • what you see in the window

• computation in coordinate system used to draw previous frame is simple:
  • incremental change \( l \) to current \( C \)
  • at time \( k \), want \( p' = l_k l_{k-1} l_{k-2} l_{k-3} ... l_5 l_4 l_3 l_2 l_1 C p \)

• each time we just want to premultiply by new matrix
  • \( p' = l C p \)
  • but we know that OpenGL only supports postmultiply by new matrix
    • \( p' = C l p \)
Viewing: Incremental Relative Motion

- sneaky trick: OpenGL modelview matrix has the info we want!
  - dump out modelview matrix with glGetDoublev()
    - C = current camera coordinate matrix
  - wipe the matrix stack with glLoadIdentity()
  - apply incremental update matrix I
  - apply current camera coord matrix C
- must leave the modelview matrix unchanged by object transformations after your display call
  - use push/pop
- using OpenGL for storage and calculation
  - querying pipeline is expensive
    - but safe to do just once per frame
Caution: OpenGL Matrix Storage

• OpenGL internal matrix storage is columnwise, not rowwise
  
a e i m
b f j n
c g k o
d h l p

• opposite of standard C/C++/Java convention
• possibly confusing if you look at the matrix from glGetUniformLocation()!
Viewing: Virtual Trackball

• interface for spinning objects around
  • drag mouse to control rotation of view volume
    • orbit/spin metaphor
    • vs. flying/driving
• rolling glass trackball
  • center at screen origin, surrounds world
  • hemisphere “sticks up” in z, out of screen
  • rotate ball = spin world
Virtual Trackball

- know screen click: \((x, 0, z)\)
- want to infer point on trackball: \((x, y, z)\)
  - ball is unit sphere, so \(||x, y, z|| = 1.0\)
  - solve for \(y\)
Trackball Rotation

- correspondence:
  - moving point on plane from \((x, 0, z)\) to \((a, 0, c)\)
  - moving point on ball from \(p_1 = (x, y, z)\) to \(p_2 = (a, b, c)\)
- correspondence:
  - translating mouse from \(p_1\) (mouse down) to \(p_2\) (mouse up)
  - rotating about the axis \(n = p_1 \times p_2\)
Trackball Computation

- user defines two points
  - place where first clicked $p_1 = (x, y, z)$
  - place where released $p_2 = (a, b, c)$
- create plane from vectors between points, origin
  - axis of rotation is plane normal: cross product
    - $(p_1 - o) \times (p_2 - o)$: $p_1 \times p_2$ if origin = $(0,0,0)$
  - amount of rotation depends on angle between lines
    - $p_1 \cdot p_2 = |p_1| |p_2| \cos \theta$
    - $|p_1 \times p_2| = |p_1| |p_2| \sin \theta$
- compute rotation matrix, use to rotate world
Picking
Reading

• Red Book
  • Selection and Feedback Chapter
    • all
  • Now That You Know Chapter
    • only Object Selection Using the Back Buffer
Interactive Object Selection

- move cursor over object, click
  - how to decide what is below?
  - inverse of rendering pipeline flow
    - from pixel back up to object
- ambiguity
  - many 3D world objects map to same 2D point
- four common approaches
  - manual ray intersection
  - bounding extents
  - backbuffer color coding
  - selection region with hit list
Manual Ray Intersection

• do all computation at application level
  • map selection point to a ray
  • intersect ray with all objects in scene.

• advantages
  • no library dependence

• disadvantages
  • difficult to program
  • slow: work to do depends on total number and complexity of objects in scene
Bounding Extents

• keep track of axis-aligned bounding rectangles

• advantages
  • conceptually simple
  • easy to keep track of boxes in world space
Bounding Extents

- disadvantages
  - low precision
  - must keep track of object-rectangle relationship
- extensions
  - do more sophisticated bound bookkeeping
    - first level: box check.
    - second level: object check
Backbuffer Color Coding

- use backbuffer for picking
  - create image as computational entity
  - never displayed to user
- redraw all objects in backbuffer
  - turn off shading calculations
  - set unique color for each pickable object
    - store in table
  - read back pixel at cursor location
    - check against table
Backbuffer Color Coding

- **advantages**
  - conceptually simple
  - variable precision

- **disadvantages**
  - introduce 2x redraw delay
  - backbuffer readback very slow
Backbuffer Example

```c
for(int i = 0; i < 2; i++)
    for(int j = 0; j < 2; j++) {
        glColor3f(1.0, 1.0, 1.0);
        for(int i = 0; i < 2; i++)
            for(int j = 0; j < 2; j++) {
                glColor3f(1.0, 1.0, 1.0);
                glBegin(GL_TRIANGLES);
                    glVertex3f(i*3.0,0,-j * 3.0);
                    glVertex3f(i*3.0,3.0,-j * 3.0);
                    glVertex3f(i*3.0,-3.0,-j * 3.0);
                glEnd();
            }  
        glEnd();
    }

http://www.lighthouse3d.com/opengl/picking/
```
Select/Hit

- use small region around cursor for viewport
- assign per-object integer keys (names)
- redraw in special mode
- store hit list of objects in region
- examine hit list

- OpenGL support
Viewport

- small rectangle around cursor
  - change coord sys so fills viewport

- why rectangle instead of point?
  - people aren’t great at positioning mouse
    - Fitts’ Law: time to acquire a target is function of the distance to and size of the target
  - allow several pixels of slop
Viewport

• nontrivial to compute
  • invert viewport matrix, set up new orthogonal projection

• simple utility command
  • gluPickMatrix(x,y,w,h,viewport)
    • x,y: cursor point
    • w,h: sensitivity/slop (in pixels)
  • push old setup first, so can pop it later
Render Modes

• `glRenderMode(mode)`
  
  • `GL_RENDER`: normal color buffer
    • default
  
  • `GL_SELECT`: selection mode for picking
  
  • `(GL_FEEDBACK): report objects drawn`
Name Stack

• again, "names" are just integers
  glInitNames()

• flat list
  glLoadName(name)

• or hierarchy supported by stack
  glPushName(name), glPopName
  • can have multiple names per object
Hierarchical Names Example

```c
for(int i = 0; i < 2; i++) {
    glPushName(i);
    for(int j = 0; j < 2; j++) {
        glPushMatrix();
        glPushName(j);
        glTranslatef(i*10.0,0,j * 10.0);
        glPushName(HEAD);
        glCallList(snowManHeadDL);
        glLoadName(BODY);
        glCallList(snowManBodyDL);
        glPopName();
        glPopName();
    }
    glPopMatrix();
}
glPopName();
```
Hit List

- `glSelectBuffer(buffersize, *buffer)`
  - where to store hit list data
- on hit, copy entire contents of name stack to output buffer.
- hit record
  - number of names on stack
  - minimum and minimum depth of object vertices
    - depth lies in the NDC z range [0,1]
    - format: multiplied by $2^{32} - 1$ then rounded to nearest int
Integrated vs. Separate Pick Function

• integrate: use same function to draw and pick
  • simpler to code
  • name stack commands ignored in render mode
• separate: customize functions for each
  • potentially more efficient
  • can avoid drawing unpickable objects
Select/Hit

• advantages
  • faster
    • OpenGL support means hardware acceleration
    • avoid shading overhead
  • flexible precision
    • size of region controllable
  • flexible architecture
    • custom code possible, e.g. guaranteed frame rate

• disadvantages
  • more complex
Hybrid Picking

- select/hit approach: fast, coarse
  - object-level granularity
- manual ray intersection: slow, precise
  - exact intersection point
- hybrid: both speed and precision
  - use select/hit to find object
  - then intersect ray with that object
OpenGL Precision Picking Hints

- `gluUnproject`
  - transform window coordinates to object coordinates given current projection and modelview matrices
  - use to create ray into scene from cursor location
  - call `gluUnProject` twice with same (x,y) mouse location
    - z = near: (x,y,0)
    - z = far: (x,y,1)
    - subtract near result from far result to get direction vector for ray
  - use this ray for line/polygon intersection
Vision/Color
Reading for Color

- RB Chap Color
- FCG Sections 3.2-3.3
- FCG Chap 20 Color
- FCG Chap 21.2.2 Visual Perception (Color)
RGB Color

• triple \((r, g, b)\) represents colors with amount of red, green, and blue
  • hardware-centric
  • used by OpenGL
Alpha

- fourth component for transparency
  - \((r,g,b,\alpha)\)
- fraction we can see through
  - \(c = \alpha c_f + (1-\alpha)c_b\)
- more on compositing later
Additive vs. Subtractive Colors

- **additive**: light
  - monitors, LCDs
  - RGB model
- **subtractive**: pigment
  - printers
  - CMY model
  - dyes absorb light

\[
\begin{bmatrix}
C \\
M \\
Y
\end{bmatrix} =
\begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix} -
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]
Component Color

• component-wise multiplication of colors
  • \((a_0, a_1, a_2) \times (b_0, b_1, b_2) = (a_0b_0, a_1b_1, a_2b_2)\)

• why does this work?
  • must dive into light, human vision, color spaces
Basics Of Color

• elements of color:
Basics of Color

• physics
  • illumination
    • electromagnetic spectra
  • reflection
    • material properties
    • surface geometry and microgeometry
      • polished versus matte versus brushed

• perception
  • physiology and neurophysiology
  • perceptual psychology
Light Sources

• common light sources differ in kind of spectrum they emit:
  • continuous spectrum
    • energy is emitted at all wavelengths
      • blackbody radiation
      • tungsten light bulbs
      • certain fluorescent lights
      • sunlight
      • electrical arcs
  • line spectrum
    • energy is emitted at certain discrete frequencies
Blackbody Radiation

• black body
  • dark material, so that reflection can be neglected
  • spectrum of emitted light changes with temperature
    • this is the origin of the term “color temperature”
      • e.g. when setting a white point for your monitor
  • cold: mostly infrared
  • hot: reddish
  • very hot: bluish
• demo:

http://www.mhhe.com/physsci/astronomy/applets/Blackbody/frame.html
Electromagnetic Spectrum

- AM radio
- FM radio, TV
- Microwave
- Ultraviolet
- Infrared
- Gamma rays
- X-rays
Electromagnetic Spectrum
White Light

- sun or light bulbs emit all frequencies within visible range to produce what we perceive as "white light"
Sunlight Spectrum

• spectral distribution: power vs. wavelength
Continuous Spectrum

- sunlight
- various “daylight” lamps
Line Spectrum

- ionized gases
- lasers
- some fluorescent lamps
White Light and Color

- when white light is incident upon an object, some frequencies are reflected and some are absorbed by the object
- combination of frequencies present in the reflected light that determines what we perceive as the color of the object
Hue

• hue (or simply, "color") is dominant wavelength/frequency

• integration of energy for all visible wavelengths is proportional to intensity of color
Saturation or Purity of Light

- how washed out or how pure the color of the light appears
  - contribution of dominant light vs. other frequencies producing white light
  - saturation: how far is color from grey
    - pink is less saturated than red
    - sky blue is less saturated than royal blue

![Energy vs Frequency Graphs](image)
Intensity vs. Brightness

• intensity: physical term
  • measured radiant energy emitted per unit of time, per unit solid angle, and per unit projected area of the source (related to the luminance of the source)

• lightness/brightness: perceived intensity of light
  • nonlinear
Perceptual vs. Colorimetric Terms

**Perceptual**

- Hue
- Saturation
- Lightness
  - reflecting objects
- Brightness
  - light sources

**Colorimetric**

- Dominant wavelength
- Excitation purity
- Luminance