Daniel Brown: guest lecture, later
Administrivia

• Hw 1 out

• Practice Hw 1 out with solutions available

• Project 1a under grading

• Review Project 1b Thursday
Examples

• Rotate moon around Earth around sun (multiple motions)

• Orient cylinder sections of 3D helix
Octave Online – working through examples

- Good for doing examples, verifying equations
- Vectors, Matrices, operations
- Open source version of Matlab
- Can also use app
- Or link Octave fcns externally to C or other languages
Back to orthogonal projection

**Orthogonal projection:** Given a vector $\vec{u}$ and a nonzero vector $\vec{v}$, it is often convenient to decompose $\vec{u}$ into the sum of two vectors $\vec{u} = \vec{u}_1 + \vec{u}_2$, such that $\vec{u}_1$ is parallel to $\vec{v}$ and $\vec{u}_2$ is orthogonal to $\vec{v}$.

$$\vec{u}_1 \leftarrow \frac{(\vec{u} \cdot \vec{v})}{(\vec{v} \cdot \vec{v})} \vec{v}, \quad \vec{u}_2 \leftarrow \vec{u} - \vec{u}_1.$$
Big idea – frame of reference

Global or local coordinate system in which to define pts and vectors

• 2D

• 3D
Understand: work through examples

• Start with obvious example
• $\mathbf{u} = <1,1>$
• $\mathbf{v} = <1,0>$

$\mathbf{u}_1 \leftarrow \frac{(\mathbf{u} \cdot \mathbf{v})}{(\mathbf{u} \cdot \mathbf{v})} \mathbf{v}$,  
$\mathbf{u}_2 \leftarrow \mathbf{u} - \mathbf{u}_1$
Understand: work through examples

• Start with obvious example
• \( u = <1,1> \)
• \( v = <1,0> \)
• \( u_1 = \frac{1}{1} \times <1,0> \)
• \( u_2 = <1,1> - <1,0> = <0,1> \)

\( u \) projects onto \( <1,0>, <0,1> \)
Understand: work through examples

- Work slowly to complex
- \( u = <0,1> \)
- \( v = <1,1> \)

\[
\vec{u}_1 \leftarrow \frac{(\vec{u} \cdot \vec{v})}{(\vec{u} \cdot \vec{v})} \vec{v}, \quad \vec{u}_2 \leftarrow \vec{u} - \vec{u}_1
\]
Understand: work through examples

- Work slowly to complex
- \( u = <0,1> \)
- \( v = <1,1> \)

- \( u_1 = (u \cdot v)/(v \cdot v) \cdot v \)
  \[ = \frac{1}{2} <1,1> = < \frac{1}{2}, \frac{1}{2}> \]

- \( u_2 = u - u_1 = <0,1> - < \frac{1}{2}, \frac{1}{2}> \)
  \[ = < - \frac{1}{2}, \frac{1}{2}> \]
Observation: are $u_1$, $u_2$ normal vectors?

- $u_1 = \langle \frac{1}{2}, \frac{1}{2} \rangle$
- $u_2 = \langle -\frac{1}{2}, \frac{1}{2} \rangle$

$\vec{u}_1 \leftarrow \frac{(\vec{u} \cdot \vec{v})}{(\vec{u} \cdot \vec{v})} \vec{v}, \quad \vec{u}_2 \leftarrow \vec{u} - \vec{u}_1$

$u_2$?

$u = \langle 0, 1 \rangle \quad v = \langle 1, 1 \rangle$
Observation: are $u_1$, $u_2$ normal vectors?

- $u_1 = \langle \frac{1}{2}, \frac{1}{2} \rangle$
- $u_2 = \langle -\frac{1}{2}, \frac{1}{2} \rangle$

$$|u_1| = \sqrt{\frac{1}{4} + \frac{1}{4}} = \sqrt{\frac{1}{2}}$$

\[ u_1 = \frac{\langle \frac{1}{2}, \frac{1}{2} \rangle}{\sqrt{\frac{1}{2}}} = \langle \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2} \rangle \]

\[ u_1 = \langle \frac{1}{2}, \frac{1}{2} \rangle / \sqrt{\frac{1}{2}} \]

\[ = \langle \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2} \rangle \]

NO

\[ \vec{u}_1 \leftarrow \frac{(u \cdot \vec{v})}{(u \cdot \vec{v})} \vec{v}, \quad \vec{u}_2 \leftarrow \vec{u} - \vec{u}_1 \]

$\vec{u}_1$? $u=0\rangle \quad v=1,1\rangle$
Problem: Ray – circle intersection

• Does the ray defined by \( \mathbf{p} \) and \( \mathbf{v} \) intersect the circle defined by \( \mathbf{c} \) and \( r \)?
Ray – circle intersection

• Does the ray defined by \( \mathbf{p} \) and \( \mathbf{v} \) intersect the circle defined by \( \mathbf{c} \) and \( r \)?

• Solutions?
A) Do equations \( \mathbf{p}(t) = \mathbf{p} + t\mathbf{v} \) and \((x-xc)^2 + (y-yc)^2 = r^2\) have solution?
B) Is sine of angle \( \times \) length to circle less than radius?
C) Length of projection of normal less than radius?
Given vectors \( u, v, \) and \( w, \) all of type Vector3, the following operators are supported:

\[
\begin{align*}
  u &= v + w; & \text{// vector addition} \\
  u &= v - w; & \text{// vector subtraction} \\
  \text{if } (u == v || u != w) \{ \ldots \} & \text{// vector comparison} \\
  u &= v \times 2.0f; & \text{// scalar multiplication} \\
  v &= w / 2.0f; & \text{// scalar division}
\end{align*}
\]

You can access the components of a Vector3 using as either using axis names, such as, \( u.x, u.y, \) and \( u.z, \) or through indexing, such as \( u[0], u[1], \) and \( u[2]. \)

The Vector3 class also has the following members and static functions.

\[
\begin{align*}
  \text{float } x &= v.magnitude; & \text{// length of } v \\
  \text{Vector3 } u &= v.normalize; & \text{// unit vector in } v\text{'s direction} \\
  \text{float } a &= \text{Vector3.Angle } (u, v); & \text{// angle (degrees) between } u \text{ and } v \\
  \text{float } b &= \text{Vector3.Dot } (u, v); & \text{// dot product between } u \text{ and } v \\
  \text{Vector3 } u1 &= \text{Vector3.Project } (u, v); & \text{// orthog proj of } u \text{ onto } v \\
  \text{Vector3 } u2 &= \text{Vector3.ProjectOnPlane } (u, v); & \text{// orthogonal complement}
\end{align*}
\]

Some of the Vector3 functions apply when the objects are interpreted as points. Let \( p \) and \( q \) be points declared to be of type Vector3. The function Vector3.Lerp is short for linear interpolation. It is essentially a two-point special case of a convex combination. (The combination parameter is assumed to lie between 0 and 1.)

\[
\begin{align*}
  \text{float } b &= \text{Vector3.Distance } (p, q); & \text{// distance between } p \text{ and } q \\
  \text{Vector3 } \text{midpoint } &= \text{Vector3.Lerp}(p, q, 0.5f); & \text{// convex combination}
\end{align*}
\]
Instant Hw1 – Ray – circle intersection

• Does the ray defined by \( \mathbf{p} \) and \( \mathbf{v} \) intersect the circle defined by \( \mathbf{c} \) and \( \mathbf{r} \)?

C) Length of projection of normal less than radius?
1) Compute \( \mathbf{v}_{\text{perp}} \)
2) Normalize \( \mathbf{v}_{\text{perp}} \)
3) Length of projection: \( \mathbf{PC} \cdot \mathbf{v}_{\text{perp}} \)
4) Is \( \mathbf{PC} \cdot \mathbf{v}_{\text{perp}} < \mathbf{r} \) ?
Moving to 3D – frame of reference

- Left handed system XYZ
Moving to 3D – frame of reference

- In Unity – (right, up, forward)
- Forward – moving forward
- Up – a sense of gravity
- Right – turn direction
Applying cross product

• Computing normal vector
  • To triangle
  • To plane

• Computing local 3D orthonormal basis

• Point-normal form of plane
  • $n \cdot (p-v0) = 0$ means $p$ is on the plane
Homogeneous coordinates: points

• Step 2: Add origin to sum

\[ p = \alpha_0 \vec{u}_0 + \alpha_1 \vec{u}_1 + O \]

• Now
  • point = \(<x, y, 1>\)
  • vector = \(<x, y, 0>\)
Affine transformations

• Key: translation, rotation, scale
Scaling

• Coordinate free - uniform scale $s$
  \[ \nu = su \]

• Coordinate based
  \[ <\nu_x, \nu_y, \nu_z> = <su_x, su_y, su_z> \]

• Scaling sizes and moves

\[ s = 2 \]
Scaling

• Coordinate free – uniform scale $s$
  \[ v = su \]

• Coordinate based
  \[ \langle v_x, v_y, v_z \rangle = \langle su_x, su_y, su_z \rangle \]

• Homogeneous coordinates – vector
  \[ \langle v_x, v_y, v_z, 0 \rangle = \langle su_x, su_y, su_z, 0 \rangle \]

• Homogeneous coordinates – points (simple scalar * doesn't work)
  \[ (v_x, v_y, v_z, 1) = (su_x, su_y, su_z, s) \]

• Scaling sizes and moves

scaling triangle with s = 2
Scaling

• Matrix form 2D
  \[ v^t = M_s u^t \]
  
  \[ M_s = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

• Vector
  \[ \langle v_x, v_y, 0 \rangle = \langle su_x, su_y, 1 * 0 \rangle \]

• Point
  \[ (q_x, q_y, 1) = \langle sp_x, sp_y, 1 * 1 \rangle \]

• Matrix multiplication on the right with transpose of vector \( v^t \)

• Works for vectors and points

• Maintains homogeneous coordinate \( w \)
Translation

• Matrix form 2D
• 
\[ v = M_t u \]
\[ M_t = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \]

• Translate point

\[ (q_x, q_y, 1) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} \]

\[ (q_x, q_y, 1) = (p_x + t_x, p_y + t_y, 1) \]
First version: coordinate based equations

- Translation by $v$: $q = p + T(v)$  
  Add vector $v$
- Scale by $a$: $q = a \ p$  
  Multiply by scalar $a$
- Rotate by $t$: $(qx, qy) = <px \cos(t) - py \sin(t), px \sin(t) + py \cos(t)>$

- Repeated scalings and translations:

  - $q = a \ (p + T(V)) = a \ (a \ p + T(V)) + T(v)) = \text{and so on ...}$

- Complex
Second version: Homogeneous coordinates

• Unify all transformations in matrix notation

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 0 & tx \\
0 & 1 & 0 & ty \\
0 & 0 & 1 & tz \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
sx & 0 & 0 & 0 \\
0 & sy & 0 & 0 \\
0 & 0 & sz & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

Identity Matrix  \hspace{1cm} \text{glTranslatef}(tx,ty,tz)  \hspace{1cm} \text{glScalef}(sx,sy,sz)

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & \cos(d) & -\sin(d) & 0 \\
0 & \sin(d) & \cos(d) & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
\cos(d) & 0 & \sin(d) & 0 \\
0 & 1 & 0 & 0 \\
-\sin(d) & 0 & \cos(d) & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
\cos(d) & -\sin(d) & 0 & 0 \\
\sin(d) & \cos(d) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

\text{glRotatef}(d,1,0,0) \hspace{1cm} \text{glRotatef}(d,0,1,0) \hspace{1cm} \text{glRotatef}(d,0,0,1)
Defining rotations

• Euler angles
  - Roll – around forward direction
  - Pitch – around right direction
  - Yaw – around up direction

• Angle Axis

• Quaternions

• In Unity
  - `transform.Rotate(x, y, z)` - Euler angles in order x,y,z
Defining rotations

- Euler angles
  - Roll – around forward direction
- Angle Axis
  - Pitch – around right direction
- Quaternions
  - Yaw – around up direction

In Unity
- `transform.Rotate(x, y, z)` - Euler angles in order x,y,z
Defining rotations

• Angle Axis

**Quaternion.AngleAxis**

```csharp
public static Quaternion AngleAxis(float angle, Vector3 axis);
```

**Description**

Creates a rotation which rotates `angle` degrees around `axis`.

```csharp
using UnityEngine;

class Example : MonoBehaviour
{
    void Start()
    {
        // Sets the transform's rotation to rotate 30 degrees around the y-axis
        transform.rotation = Quaternion.AngleAxis(30, Vector3.up);
    }
}
```
Interpolating transformations

• Translation. Easy – move v*\(dt\) each frame
• Scale. Easy – scale by s*\(dt\) each frame

• Interpolating rotations? Harder
  • Interpolate Euler angles? Doesn’t work well
  • Interpolate Axis Angle? Better
  • Interpolate Quaternions? Best Why Unity uses them.
Quaternion.Slerp

public static Quaternion Slerp(Quaternion a, Quaternion b, float t);

Description
Spherically interpolates between a and b by t. The parameter t is clamped to the range [0, 1].

// Interpolates rotation between the rotations "from" and "to"
// (Choose from and to not to be the same as
// the object you attach this script to)

using UnityEngine;
using System.Collections;

class ExampleClass : MonoBehaviour
{
    public Transform from;
    public Transform to;

    private float timeCount = 0.0f;

    void Update()
    {
        transform.rotation = Quaternion.Slerp(from.rotation, to.rotation, timeCount);
        timeCount += Time.deltaTime;
    }
}
Defining rotations

- Angle Axis

**Quaternion.AngleAxis**

```csharp
public static Quaternion AngleAxis(float angle, Vector3 axis);
```

**Description**

Creates a rotation which rotates `angle` degrees around `axis`.

```csharp
using UnityEngine;

public class Example : MonoBehaviour
{
    void Start()
    {
        // Sets the transforms rotation to rotate 30 degrees around the y-axis
        transform.rotation = Quaternion.AngleAxis(30, Vector3.up);
    }
}
```
Interpolating transformations

• Translation. Easy – move v*dt each frame
• Scale. Easy – scale by s*dt each frame

• Interpolating rotations? Harder
  • Interpolate Euler angles? Doesn’t work well
  • Interpolate Axis Angle? Better
  • Interpolate Quaternions? Best

Why Unity uses them.
Quaternion.Slerp

public static Quaternion Slerp(Quaternion a, Quaternion b, float t);

Description

Spherically interpolates between a and b by t. The parameter t is clamped to the range [0, 1].

// Interpolates rotation between the rotations "from" and "to"
// (Choose from and to not to be the same as
// the object you attach this script to)

using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour
{
    public Transform from;
    public Transform to;

    private float timeCount = 0.0f;

    void Update()
    {
        transform.rotation = Quaternion.Slerp(from.rotation, to.rotation, timeCount);
        timeCount = timeCount + Time.deltaTime;
    }
}
Readings

• David Mount's lectures on Geometry and Geometric Programming