Still at tables ...
Administrivia

• Instant Hw1 due

• Project 1a under grading

• Review Project 1b Thursday

• Full Hw1 coming soon
Today’s question

Computing AND changing distances, directions and orientations
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 \( \vec{v} \).

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

2D frame of reference
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
• \( u = \langle 1, 1 \rangle \)
• \( v = \langle 1, 0 \rangle \)

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

- Start with obvious example
- \( u = \langle 1, 1 \rangle \)
- \( v = \langle 1, 0 \rangle \)

- \( u_1 = 1/1 \times \langle 1, 0 \rangle \)
- \( u_2 = \langle 1, 1 \rangle - \langle 1, 0 \rangle = \langle 0, 1 \rangle \)

\( u \) projects onto \( \langle 1, 0 \rangle, \langle 0, 1 \rangle \)

\[ \begin{align*}
\vec{u}_1 & \leftarrow \frac{(\vec{u} \cdot \vec{v})}{(\vec{u} \cdot \vec{v})} \vec{v}, \\
\vec{u}_2 & \leftarrow \vec{u} - \vec{u}_1
\end{align*} \]
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
• $u = <0,1>$
• $v = <1,1>$

• $u_1 = (u \cdot v)/(v \cdot v) \cdot v$
  $= \frac{1}{2} <1,1> = < \frac{1}{2}, \frac{1}{2} >$

• $u_2 = u - u_1 = <0,1> - < \frac{1}{2}, \frac{1}{2} >$
  $= <- \frac{1}{2}, \frac{1}{2} >$

$\vec{u}_1 \leftarrow \frac{(\vec{u} \cdot \vec{v})}{(\vec{u} \cdot \vec{v})} \cdot \vec{v}, \quad \vec{u}_2 \leftarrow \vec{u} - \vec{u}_1$
Observation: are $u_1$, $u_2$ normal vectors?

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

$$
\vec{u}_1 \leftarrow \frac{(\vec{u} \cdot \vec{v})}{(\vec{u} \cdot \vec{v})} \vec{v}, \quad \vec{u}_2 \leftarrow \vec{u} - \vec{u}_1
$$
Observation: are u1, u2 normal vectors?

- $u_1 = \left< \frac{1}{2}, \frac{1}{2} \right>$
- $u_2 = \left< -\frac{1}{2}, \frac{1}{2} \right>$
- $|u_1| = \sqrt{\frac{1}{4} + \frac{1}{4}} = \sqrt{\frac{1}{2}}$
  
  $u_1 = \frac{\left< \frac{1}{2}, \frac{1}{2} \right>}{\sqrt{1/2}} = \left< \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2} \right>$

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

$v = \left< 1, 1 \right>$  
$u = \left< 0, 1 \right>$

NO
Chalkboard work – solving with perp vector

- Using perp vector to create orthonormal basis
- Not just orthogonal

- Ortho – at right angles
- Normal – each vector is unit length

- Orthonormal basis gives us local frame of reference
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
Instant Hw1 – Ray – circle intersection

• Does the ray defined by $p$ and $v$ intersect the circle defined by $c$ and $r$?
Instant Hw1 – Ray – circle intersection

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

• Answers:
  A) Do equations $p(t) = p + tv$ and $(x-\mathbf{c})^2 + (y-\mathbf{c})^2 = r^2$ have solution?
  B) Is sine of angle * length to circle less than radius?
  C) Length of projection of normal less than radius?
Instant Hw1 – Ray – circle intersection

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

C) Length of projection of normal less than radius?
   1) Compute \( v_{\text{perp}} \)
   2) Normalize \( v_{\text{perp}} \)
   3) Distance center to line: \( PC \cdot v_{\text{perp}} \)
   4) Is \( PC \cdot v_{\text{perp}} < r \) ?
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
Working in 3D – cross product

- Cross product of two vectors
- Right handed system! (Unity is LHS)
Working in 3D – cross product

• Cross product of two vectors
• Right handed system! (Unity is LHS)

\[
\vec{u} \times \vec{v} = \begin{pmatrix}
    u_yv_z - u_zv_y \\
    u_zv_x - u_xv_z \\
    u_xv_y - u_yv_x
\end{pmatrix}.
\]

• Cross product is
  • Skew symmetric. \( u \times v = -v \times u \)
  • Non associative. \( (u \times v) \times w \neq u \times (v \times w) \)
  • Bilinear. \( au \times (v + w) = a(u \times v + u \times w) \)
Computing cross product

- Matrix determinant approach

\[ \vec{u} \times \vec{v} = \begin{vmatrix} \vec{e}_x & \vec{e}_y & \vec{e}_z \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix} \]

- \( \vec{e}_x = \langle 1, 0, 0 \rangle \)
- \( \vec{e}_y = \langle 0, 1, 0 \rangle \)
- \( \vec{e}_z = \langle 0, 0, 1 \rangle \)

- Will review matrix operations
Applying cross product

• Computing normal vector
  • To triangle
  • To plane

• Computing local 3D orthonormal basis

• Point-normal form of plane
  • \( \mathbf{n} \cdot (\mathbf{p} - \mathbf{v}_0) = 0 \) means \( \mathbf{p} \) is on the plane
Tiny Planet example

• Given $p$, $c$ and $q$
• Compute $f$, $u$ and $r$

Fig. 2: Tiny-planet coordinate frame.
Tiny Planet example

- Given p, c and q
- Compute f, u and r
  
  - u = normalize(p-c)
  - r = normalize((q-c) x (u))
  - f = u x r

Fig. 2: Tiny-planet coordinate frame.
Sin rule for cross products

• Relates magnitude of cross product to sin of angle and area of parallelogram

\[ |\vec{u} \times \vec{v}| = |u||v|\sin \theta. \]

• If \( a \times b = 0 \) then ...?

• If \( |a| = |b| = 1 \) and \( |a \times b| = 1 \), then ...?

• In general, the smaller \( |a \times b| \), the less numerically stable the result
Homogeneous coordinates: vectors

- Step 1: Represent vectors as linear combinations of others: \( \vec{v} = <a_0, a_1> \)

\[
\vec{v} = \alpha_0 \vec{u}_0 + \alpha_1 \vec{u}_1,
\]

- \( u_0 \) and \( u_1 \) are basis vectors
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
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 $t$: $(qx,qy) = \langle px \cos(t) - py \sin(t), \ px \sin(t) + \ py \cos(t) \rangle$

• 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

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

\begin{align*}
\text{Identity Matrix} & & \text{glTranslate(tx,ty,tz)} & & \text{glScale(sx,ty,sz)} \\
\text{glRotate(d,1,0,0)} & & \text{glRotate(d,0,1,0)} & & \text{glRotate(d,0,0,1)}
\end{align*}
Chalkboard – review all transformations
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 z, x, y
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**

```csharp
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].

```csharp
// 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;
    }
}
```
Activity 4b: Build a computer game

• At each table plan out a game for your team. Answer these questions (quickly!)

• What platform(s)?
• Any special hardware or peripherals needed?
• What software elements needed?
• Build from scratch or use engine? Which language or engine?
• What assets will you need? How will you make or get them?
Given vectors \( u, v, \) and \( w, \) all of type \( \text{Vector3}, \) 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 \( \text{Vector3} \) 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 \( \text{Vector3} \) class also has the following members and static functions.

\[
\begin{align*}
    \text{float} \ x &= v.\text{magnitude}; & \text{// length of } v \\
    \text{Vector3} \ u &= v.\text{normalize}; & \text{// unit vector in } v'\text{’s direction} \\
    \text{float} \ a &= \text{Vector3.\text{Angle}} \ (u, v); & \text{// angle (degrees) between } u \text{ and } v \\
    \text{float} \ b &= \text{Vector3.\text{Dot}} \ (u, v); & \text{// dot product between } u \text{ and } v \\
    \text{Vector3} \ u1 &= \text{Vector3.\text{Project}} \ (u, v); & \text{// orthog proj of } u \text{ onto } v \\
    \text{Vector3} \ u2 &= \text{Vector3.\text{Project0nPlane}} \ (u, v); & \text{// orthogonal complement}
\end{align*}
\]

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

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

• David Mount's lectures on Geometry and Geometric Programming