Geometry and Geometric Programming III

CMSC425.01 Spring 2019

Still at tables ...
Administrivia

• Project 1a grades released tonight

• Final project introduction this week

• Hw1 posted to web site, due next Sunday
Final project proposals

Include
• Team members
• Game title
• General description
• Platform and resources
• Coordination
Final project proposals

Include
• Team members
• Game title
• General description
• Platform and resources
• Coordination

Advice
• Teams of 2-3 best, > 4 ask
• Demoable at end of semester
• Do one thing well
• Involve entire team
• Design in layers
• K.I.S.S. (look it up ...)
Today’s question

How do we move and orient shapes?
Examples

• Rotate moon around Earth around sun (multiple motions)
• Orient cylinder sections of 3D helix
Start with frame of references

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

• 2D

• 3D
Affine transformations

- Key: translation, rotation, scale
Scaling

• Coordinate free - uniform scale $s$

\[ v = su \]

• Coordinate based

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

• 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 $\ast$ doesn't work)
  
  \[ (v_x, v_y, v_z, 1) = (su_x, su_y, su_z, s) \]

- Scaling sizes and moves
  
  $s = 2$
Scaling

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

• Vector
  \[ < v_x, v_y, 0 > = < su_x, su_y, 1 * 0 > \]

• Point
  \[ (q_x, q_y, 1) = < sp_x, sp_y, 1 * 1 > \]

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

• Works for vectors and points

• Maintains homogeneous coordinate \( w \)
Scaling – non-uniform

- Matrix form 2D

\[ v = M_s u \]

\[
M_s = \begin{bmatrix}
  s_x & 0 & 0 \\
  0 & s_y & 0 \\
  0 & 0 & 1
\end{bmatrix}
\]

\[ \text{sx} = 1 \quad \text{sy} = 2 \]
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: \( q = a \cdot p \) Multiply by scalar a

• Rotate by t: \((q_x, q_y) = <p_x \cdot \cos(t) - p_y \cdot \sin(t), p_x \cdot \sin(t) + p_y \cdot \cos(t)>\)

• Repeated scalings and translations:

\[ q = a \cdot (p + T(V)) = a \cdot ((a \cdot p + T(V)) + T(v)) = \text{and so on} \ldots \]

• Complex
Second version: Homogeneous coordinates

• Unify all transformations in matrix notation

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

Identity Matrix \quad \text{glTranslate}(tx,ty,tz) \quad \text{glScale}(sx,sy,sz)

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

\text{glRotate}(d,1,0,0) \quad \text{glRotate}(d,0,1,0) \quad \text{glRotate}(d,0,0,1)
Chalkboard – review all transformations
Defining rotations

- Euler angles
  - Roll – around forward direction
  - Pitch – around right direction
- Angle Axis
- 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:

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

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.

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

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 \( \text{Vector3.Lerp} \) is short for \textit{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.)

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

• David Mount's lectures on Geometry and Geometric Programming