Teams reminder
Finish rotation example
Sound
Upcoming projects

- Project 4 requires teams of two
  - You should have your team formed by the middle of next week.
  - No individual solutions accepted!
  - I will post a link to help uncommitted students pair up.
Transformations in XNA

- Classroom demos
  - Consider a rotating flipper again:

  ![Diagram of a rotating flipper](image)

  A reasonable representation
  (not quite rotated the same as mine)

  My sprite
Transformations in XNA

- Classroom demos
  - Consider a rotating flipper again:
    » Use circles and line segments to define boundaries
Transformations in XNA

● Classroom demos
  – Consider a rotating flipper again:
    » Define elements using a sprite origin and sprite coordinate system

» Actual world/screen coordinates irrelevant for now
Transformations in XNA

● Classroom demos
  – Consider a rotating flipper again:
    » Define elements using a sprite origin and sprite coordinate system
    
    » If using my line segment code, define line points in clockwise direction around outside of shape
Transformations in XNA

● Classroom demos
  – Consider a rotating flipper again:
    » Define elements using a sprite origin and sprite coordinate system

  (35, 5) radius=7

  » Etc.
Transformations in XNA

- Classroom demos
  - In preparation for drawing the sprite
    » Find the rotation origin in the graphic (using the graphic file’s coordinate system)

spriteOrigin = (12, 13)
Transformations in XNA

- Classroom demos
  - In preparation for drawing the sprite
    » Determine the screen location, scale (usually 1.0), and rotation (in radians) that you want to use for drawing the sprite

*Examples only:*
- screenLoc = (250, 575)
- spriteScale = 1
- spriteRot = -0.25
Transformations in XNA

- Classroom demos
  - Draw it using the overloaded SpriteBatch draw:

```csharp
spriteBatch.Draw (leftFlipperTexture,
                 spritePos, null, Color.WHITE,
                 spriteRot,
                 spriteOrigin,
                 spriteScale,
                 SpriteEffects.NONE, 0);
```
Transformations in XNA

- Computing collisions
  - Sprite defined in local coordinate system, drawn sprite may be translated, rotated, and scaled on the screen.

  » You need to convert each line or circle from sprite coordinates to screen/world coordinates
Computing collisions

» Simple: Create a transformation matrix to perform the coordinate space conversion:

```csharp
Matrix transform;
transform = Matrix.Identity;
transform = transform * Matrix.CreateRotationZ(spriteRot);
transform = transform * Matrix.CreateScale(spriteScale);
transform = transform * Matrix.CreateTranslation(spritePos.X, spritePos.Y, 0);
```
Transformations in XNA

- Computing collisions
  - Build new lines / circles by taking the sprite lines/circles and converting the coordinates
  - Transform each point using the transformation matrix to convert it to screen space.
  - Hint – you will need to convert your 2D points to 3D points to transform them, then convert them back to 2D

```csharp
Vector3 screenLoc, spriteLoc = someSpriteLoc;
screenLoc = Vector3.Transform(spriteLoc, transform);
```
Transformations in XNA

- Computing collisions
  - If your original boundaries matched your original sprite, and if you constructed the transform correctly, the converted boundaries will match the transformed sprite! Yay.

  » Note – my numbers above are guesses.
Remember to build copies of your line segments

- Don’t modify the originals, you’ll need them for the next transform.
- Don’t attempt to make incremental transforms in screen coordinate space
  » Floating point errors will quickly accumulate destroying your result
  » Always re-transform from the original sprite space to the screen/world space
How to ‘hit’ the ball with the flipper

- The ‘net’ velocity between the ball and the flipper should be used in reflections.

- Since the flipper is anchored (stationary) at one end, the velocity of the flipper varies.
How to ‘hit’ the ball with the flipper

- First, compute the point of impact.
- Second, compute the velocity at that point.
- Third, subtract it from the ball’s velocity
Point of impact for a line:

- (Remember – use screen space coordinates – use your transformation.)
**Point of impact for a line:**

- Compute vectors V and N using coordinates of line and circle
Point of impact for a line:

- Normalize N, then find out how much of V is in N’s direction. Compute D.

\[ |D| = \vec{N} \cdot V \]

\[ D = \vec{N} \times |D| \]
Point of impact for a line:

- Add D to the starting point to get the point of impact.
(Nearly) every point on the flipper is moving differently around the axis of rotation. Compute the velocity at P.

Velocity at that point:
Velocity at that point:

- Find how far P is from the axis of rotation. Call this distance k.
Velocity at that point:

- The perimeter of the circle is $2\pi k$ pixels.
Velocity at that point:

- Velocity is pixels per second. Determine your radial velocity in radians, something like \(-\pi/4\) radians/second.
Velocity at that point:

- Use unit analysis to do the rest!

2πk pixels
2π radians

π/4 radians / second
Velocity at that point:

- velocity = (pixels/radian) * (radians/second)
- velocity = pixels / second
We know how fast the point is moving, next compute the direction. Use the vector $K$ that you created to compute $k$. $|K| = k$, right?

Direction at that point:

- $2\pi k$ pixels
- $2\pi$ radians

$\pi/4$ radians / second
Direction at that point:

- $K$ is just some pair $(\Delta x, \Delta y)$. The point is moving perpendicular to $K$, either $(-\Delta y, \Delta x)$ or $(\Delta y, -\Delta x)$ depending on rotation direction.
**Velocity vector at that point:**

- $K'$ is our velocity vector for the point. Normalize it (to make it’s length 1) and multiply it by our speed.

$$K' = (\Delta y, -\Delta x)$$
Velocity vector at that point:

- Done!
- Vel vector at $P = \text{normalize}(K') \times \text{velocity}$
Issues

• What about collisions with moving circles?
  – Same basic ideas, we need the impact point, its velocity, and its normal vector.
Issues

● What about glancing blows?
  – Need to adjust for collision angle not matching normal angle. (Most of the time you won’t see this, but once in a while things will look strange.)
Issues

- What about glancing blows?
  - Simple enough – multiply the velocity vector by the cosine of the angle between velocity normal with the impact normal. (Use a dot product on normalized vectors!)
Sound in XNA

- I will do an in-class example of this
  - If you missed class, use the on-line tutorials
  - You may want to look up how to bend pitch and volume.