Collision Detection

Based on Collision Series
On XNA Creators Club

Collision Detection

- Circular
- Rectangular
- Pixel-based
- Rotated rectangles
- Pixel-based with rotations
- Minimize work (to do)

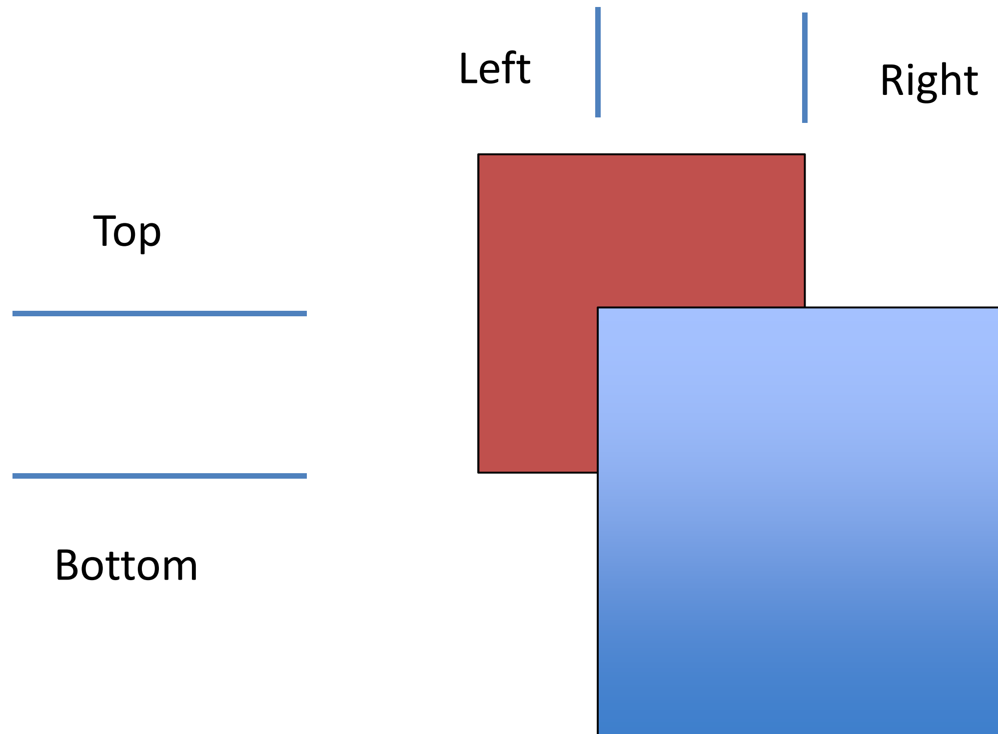
Circular Collision Detection

- Simple
- Maintain information about
  - the object's center and
  - the distance from the center to the furthest point on the object, i.e. the radius of the circle.
- Determine the distance between two objects' centers and see if it is less than the sum of their bounding circle's radii.

Rectangular Collision Detection

- Determine the “rectangle of intersection”
- If that rectangle is “degenerate” then there is no intersection, otherwise there is.
- What’s a degenerate rectangle?
- One with a height or width less than or equal to zero pixels.

Overlapping Rectangles



- Lowest Top
- Highest Bottom
- Rightmost Left
- Leftmost Right

Rectangular Collision Detection

- Compute the top of the possible rectangular intersection
  - It will be the top of the two rectangles that is lower on the screen.
- Compute the bottom of the possible rectangular intersection
  - It will be the bottom that is higher on the screen.
  - To get the bottom, we have to add the height of the sprite to the top.
- Compute the left side
  - It will be the left side that is furthest to the right.
- Compute the right side
  - It will be the right side that is furthest to the left.
- When does intersection fail?
  - If the bottom is higher up than the top
  - If the left side is further right than the right side.
- Otherwise we have a hit!!

Compute Intersection Rectangle

```
// Find the bounds of the rectangle intersection
int top = Math.Max(rectangleA.Top, rectangleB.Top);
int bottom = Math.Min(rectangleA.Bottom, rectangleB.Bottom);
int left = Math.Max(rectangleA.Left, rectangleB.Left);
int right = Math.Min(rectangleA.Right, rectangleB.Right);

// Did we intersect?
//   Note that XNA Rectangle's Bottom is Y +Height,
//   so it is not part of the rectangle.
return (top > bottom && left > right);
```

Pixel-based Collision Detection

- Rectangular collision can have annoying results
- We would prefer not to report a collision when none has occurred yet.
- Steps?
  - Get pixel data for sprites
  - Get the intersection rectangle
  - Check its pixels

Pixel Data

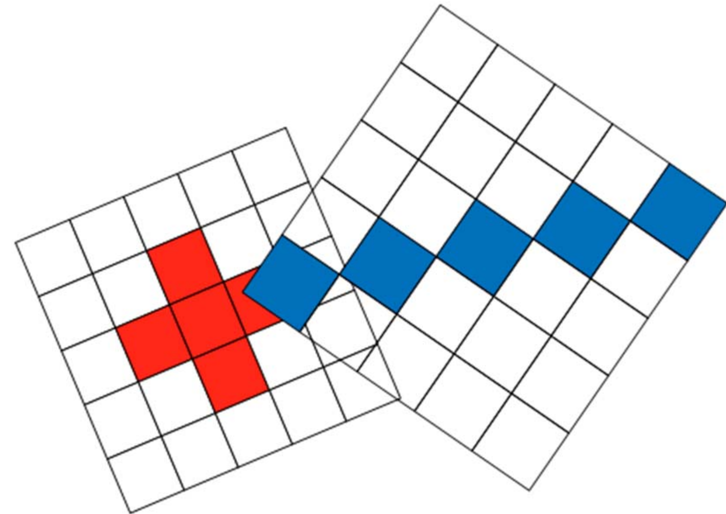
- Which pixels actually intersected?
- Compute the rectangular intersection.
- Look at the pixels from each sprite.
- XNA’s Texture2D has a GetData method that fills a one-dimensional Color array.
- Each Color has 4 properties, R, G, B and A.
- If the pixels for each shape have a non-zero alpha channel, then you have a collision!

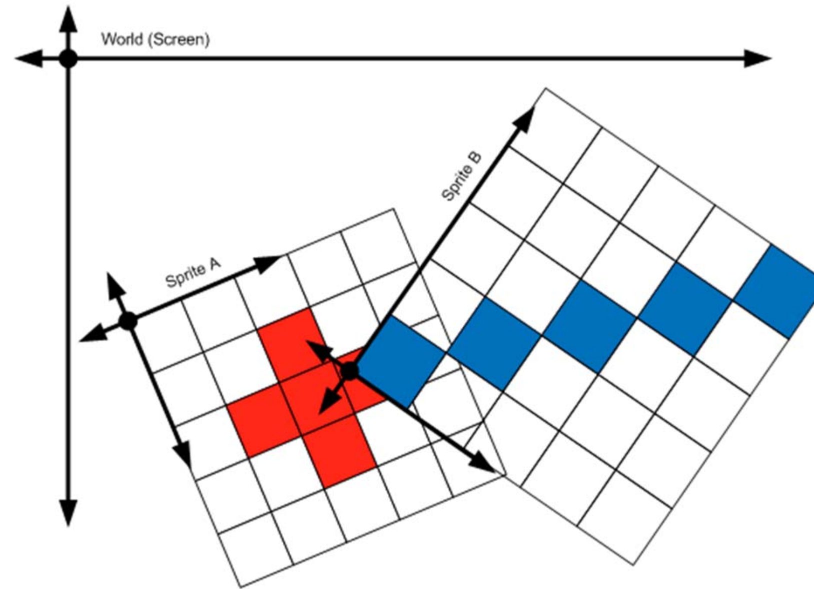
Check all Pixels for Collision

```
for (int y = top; y < bottom; y++) {     // top to bottom
    for (int x = left; x < right; x++) { // left to right
        // Get the Color array indices
        int indexA = x - rectangleA.Left
                   + (y-rectangleA.Top) * rectangleA.Width;
        int indexB = x - rectangleB.Left
                   + (y-rectangleB.Top) * rectangleB.Width;
// Get the color of both pixels at this point
        Color colorA = dataA[indexA];
        Color colorB = dataB indexB];
       // If both pixels are not completely transparent,
       // then an intersection has been found
       if (colorA.A != 0 && colorB.A != 0) return true;
    } // for x
} // for y
// No intersection found
return false;
```

Transformed Collision Detection

- What happens when the sprites have not only been translated
- But also rotated?

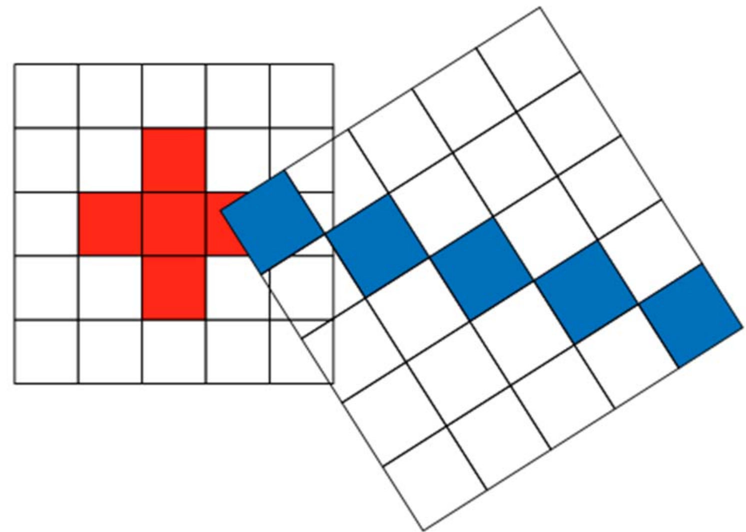




Map from A into the *world* view.
Then map to B's *local* view.

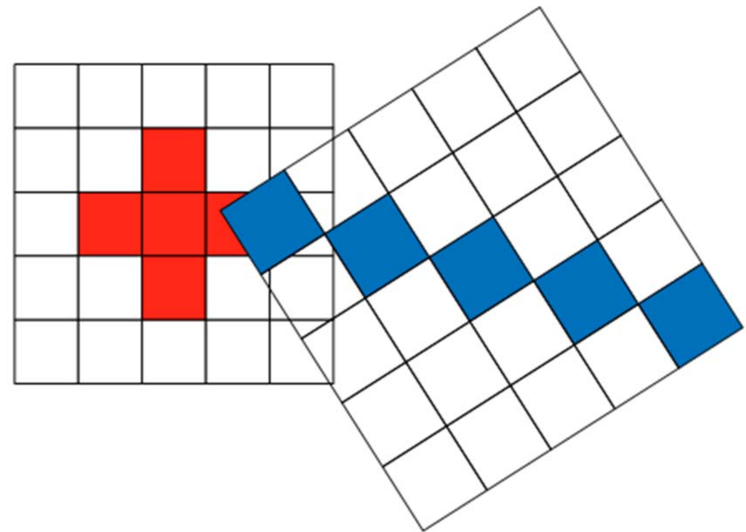
Idea

Code is simplified if we can map from “untransformed” sprite A (red) to transformed sprite B (blue)



Representing Transformations

- How can we represent transformations?
- Linear algebra!!!
- Transformations can be represented by "matrices"
- And "combined" by multiplication.
- (We'll talk about this more later)



```
Matrix transformAToB = transformA * Matrix.Invert(transformB);
```

Creating Transformations

```
// Create the person's transformation matrix each time he moves.
Matrix personTransform =
            Matrix.CreateTranslation(new Vector3(personPosition, 0.0f));

// Create the block's transform
// This is tricky!!!  (need pictures)
Matrix blockTransform =
        Matrix.CreateTranslation(new Vector3(-blockOrigin, 0.0f))
              * Matrix.CreateRotationZ(blocks[i].Rotation)
              * Matrix.CreateTranslation(newVector3(blocks[i].Position,
               0.0f));
// Check collision with person
personHit  = IntersectPixels(personTransform, personTexture.Width,
                             personTexture.Height, personTextureData,
                             blockTransform, blockTexture.Width,
                             blockTexture.Height, blockTextureData));
```

IntersectPixels

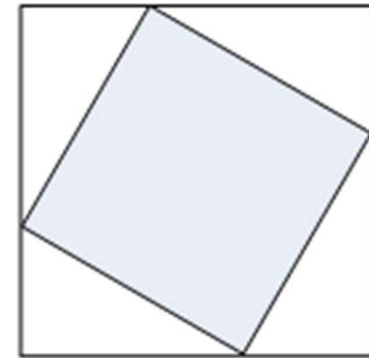
```
Matrix transformAToB = transformA * Matrix.Invert(transformB);
for (int yA = 0; yA < heightA; yA++)  {  // Looping over rows in A
    for (int xA = 0; xA < widthA; xA++)  {  // Looping over pixel in a row in A
        // Calculate this pixel's location in B
        // "transform" the A pixel into a location in B's "space"
        Vector2 positionInB =
            Vector2.Transform(new Vector2(xA, yA), transformAToB);

        // Round to the nearest pixel
        int xB = (int)Math.Round(positionInB.X);
        int yB = (int)Math.Round(positionInB.Y);

        if (0 <= xB && xB < widthB && 0 <= yB && yB < heightB)  {  // Overlap?
            Color colorA = dataA[xA + yA * widthA]; // Get A's color
            Color colorB = dataB[xB + yB * widthB];  // Get B's color
            if (colorA.A != 0 && colorB.A != 0)  {       // Both visible?
                return true;   //   intersection!
            }}}}
return false;   // No intersection.
```

First Optimization Rectangular Collision detection with Rotated Rectangles

- Doing pixel by pixel checking / transformations is expensive!
- First should do a sanity check
- Rotate *just* the corners of the sprites
- Then create a bounding box. (How? next slide)
- Finally check for rectangular intersection.



CalculateBoundingRectangle

```
// Transform all four corners into work space
Vector2.Transform(ref leftTop, ref transform, out leftTop);
Vector2.Transform(ref rightTop, ref transform, out rightTop);
Vector2.Transform(ref leftBottom, ref transform, out leftBottom);
Vector2.Transform(ref rightBottom, ref transform, out rightBottom);

// Find the minimum and maximum extents of the rectangle in world space
// Note that Vector2.Min(v1, v2) = Vector(min(v1.x, v1.y), min(v1.y, v2.y)
Vector2 min = Vector2.Min(Vector2.Min(leftTop, rightTop),
                          Vector2.Min(leftBottom, rightBottom));
Vector2 max = Vector2.Max(Vector2.Max(leftTop, rightTop),
                          Vector2.Max(leftBottom, rightBottom));
// Return as a rectangle
return new Rectangle((int)min.X, (int)min.Y,
                     (int)(max.X - min.X), (int)(max.Y - min.Y));
```

Second Optimization
Eliminate Per-Pixel Transformation

- Doing pixel by pixel transformations is expensive!
  - Yes, we said that already.
- Going across a row in A results in a *fixed* sized translation in B. Just add vectors!
- Same for going from row to row.