the gamedesigninitiative at cornell university

Lecture 14

Geometry & Texture

Graphics Lectures

- Drawing Images
  - Coordinates & Transforms
  - Images & Colors

bare minimum to draw graphics

- Drawing Perspective
  - Camera
  - Projections

side-scroller vs. top down

- **Drawing Primitives**
  - Meshes
  - Shaders

necessary for lighting & shadows

The OpenGL Conundrum

- LibGDX is built on top of **OpenGLES**
  - Mobile version of OpenGL, for targeting Android
  - Supported on desktop (Windows, macOS) also
- LibGDX uses **OpenGLES 2.0**
  - This is an ancient version **deprecated 15 years ago**!
  - LibGDX too much legacy code to support newer versions
- GDIAC extensions support **OpenGLES 3.0**
  - Most recent version of OpenGLES
  - Future platforms are likely moving to **Vulkan**

The OpenGL Conundrum

- LibGDX is built on top of **OpenGLES**
  - Mobile version of OpenGL, for targeting Android
  - Supported on desktop (Windows, macOS) also
- LibGD
  - This **o**!
  - LibG ersions
- GDIAC extensions support **OpenGLES 3.0**
  - Most recent version of OpenGLES
  - Future platforms are likely moving to **Vulkan**

But the GDIAC extensions are backward compatible

Why Does This Matter?

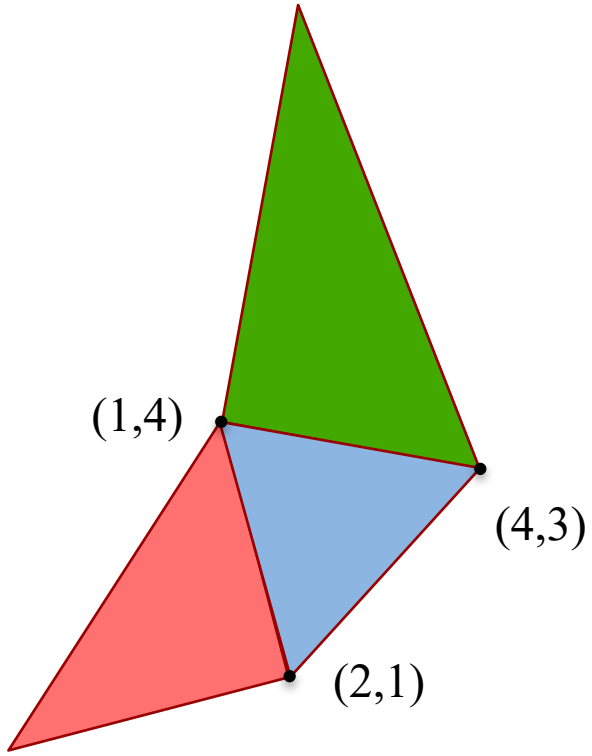
- OpenGL relies on **shader programs**
  - Instruct the graphics card what to do
  - Written in a graphics language like GLSL
  - Loaded when your game first starts
- OpenGLES 2.0 and 3.0 are **not compatible**
  - They use different dialects of GLSL
  - GDIAC supports both kinds of shaders
  - LibGDX only supports OpenGLES 2.0
- This matters if you want your own shaders

Controlling the OpenGL Version

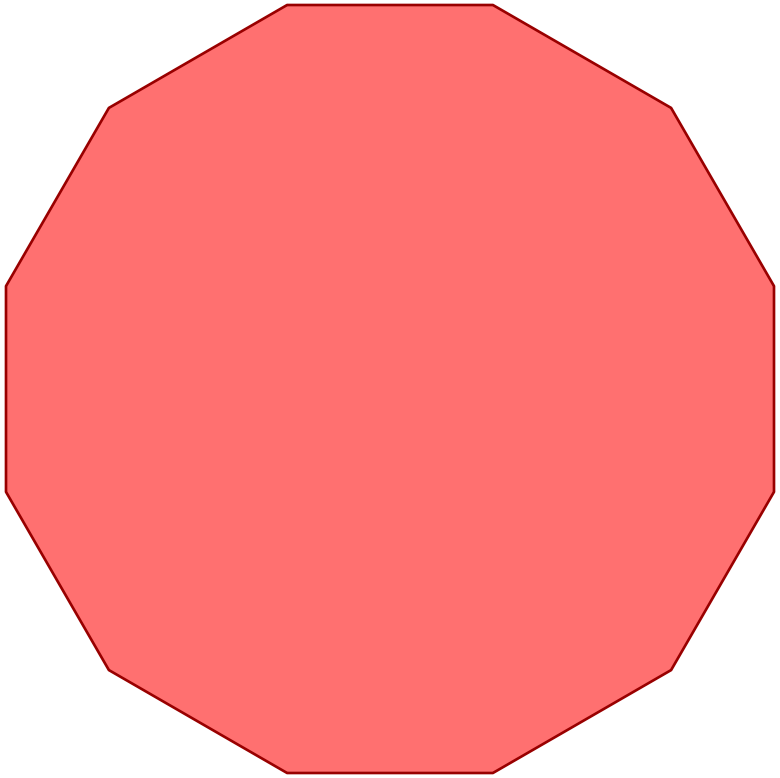
- LibGDX has **two** OpenGLES interfaces
  - `Gdx.gl20` // Never null, even if using 3.0
  - `Gdx.gl30` // Null only if using 2.0
- We set the version in `DesktopLauncher`
  - Option `config.useGL30`
  - Value is `true` by default
- You almost never need to change this
  - Only needed if using legacy LibGDX features
  - **Example:** ShapeRenderer

Drawing In OpenGL

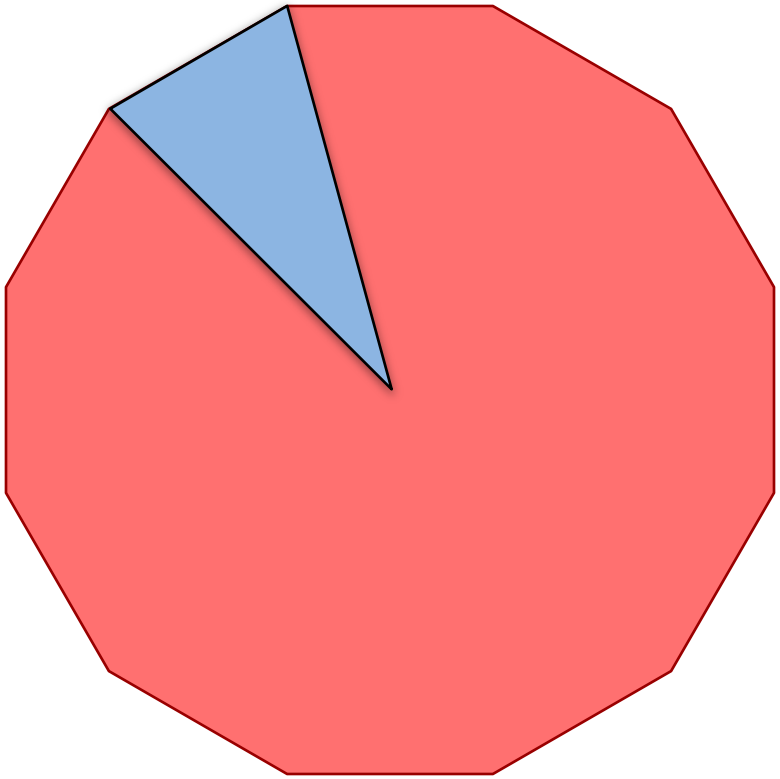
- Everything is made of **triangles**
  - Mathematically "nice"
  - Hardware support (GPUs)
- Specify with **three vertices**
  - Coordinates of corners
- Composite for complex shapes
  - Array of vertex objects
  - Each 3 vertices = triangle



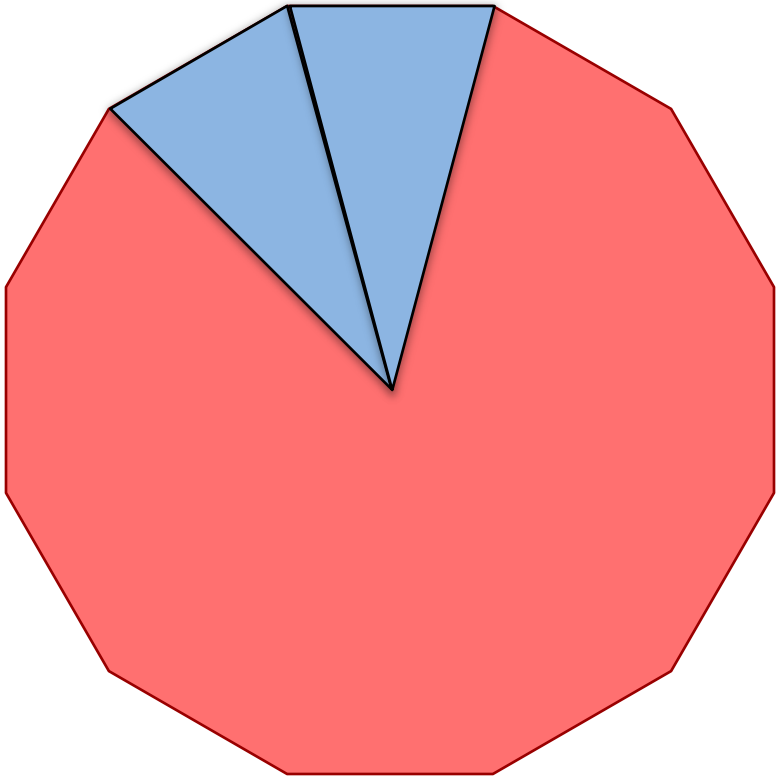
Triangulation of Polygons



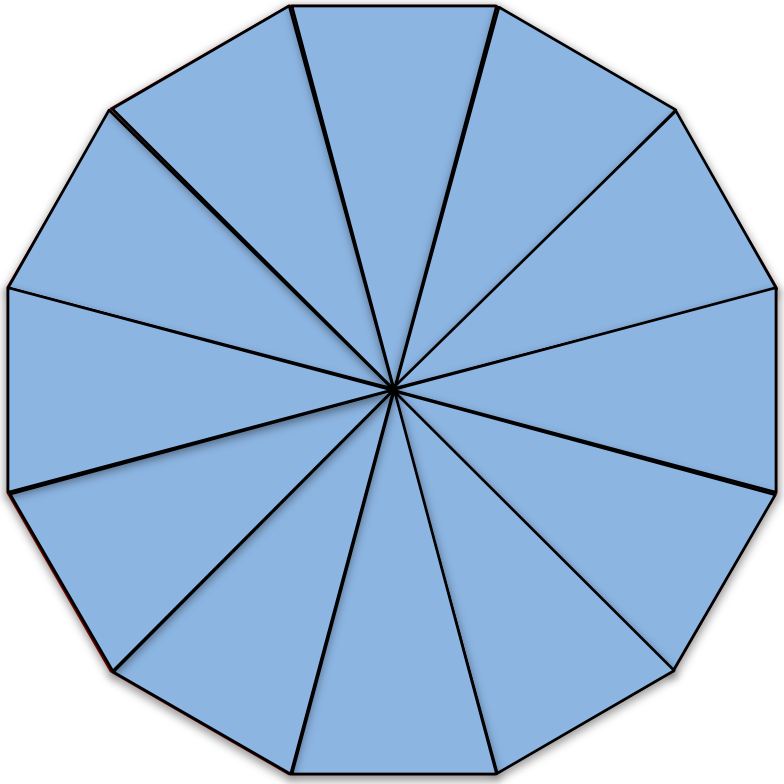
Triangulation of Polygons



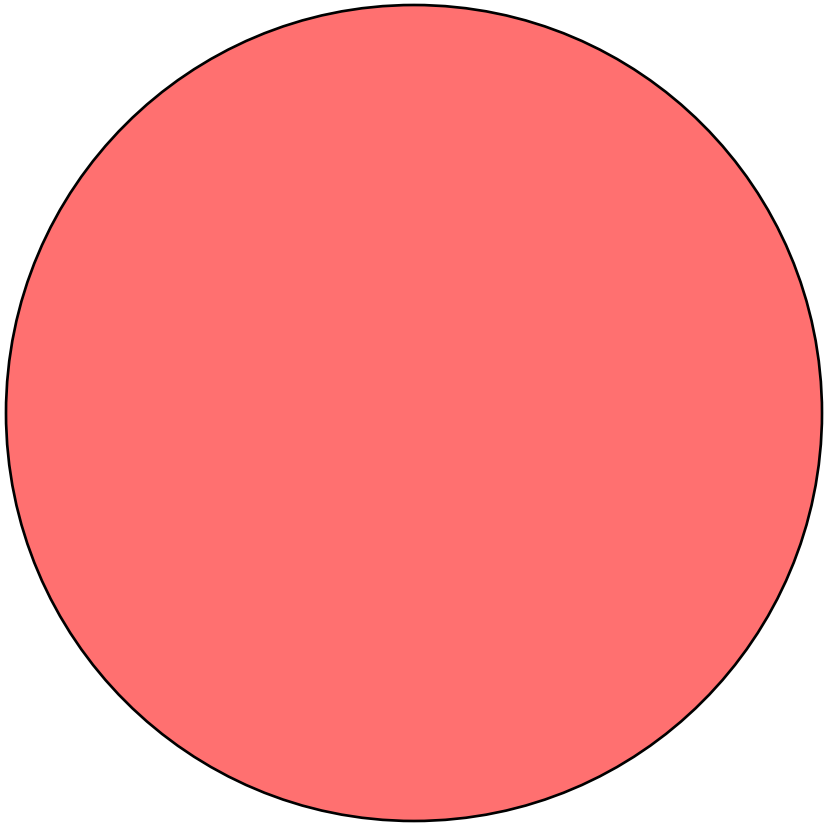
Triangulation of Polygons



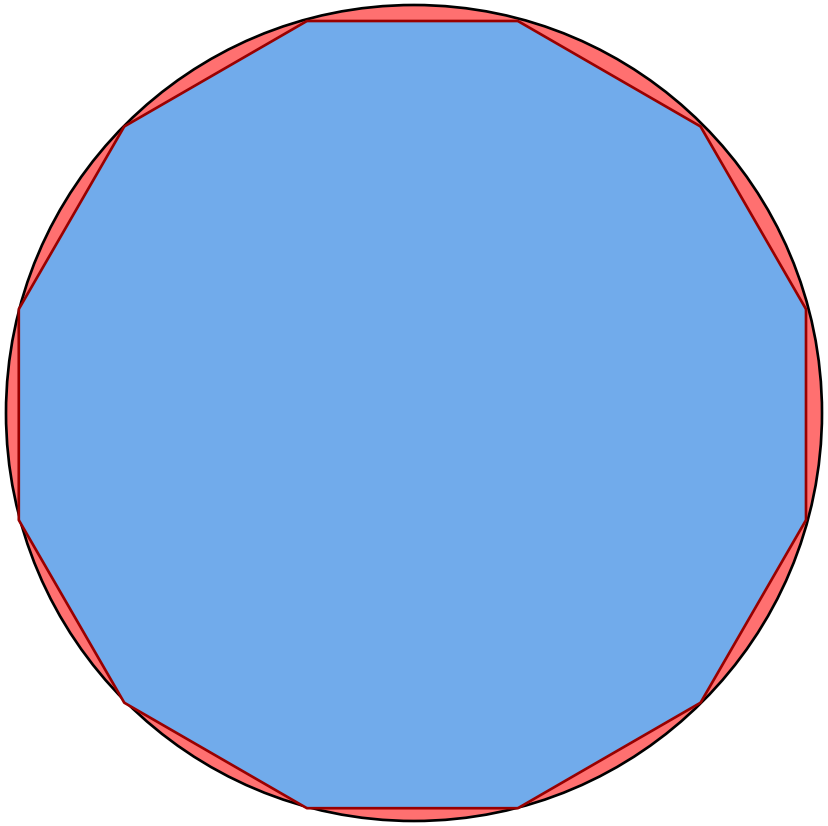
Triangulation of Polygons



Round Shapes?

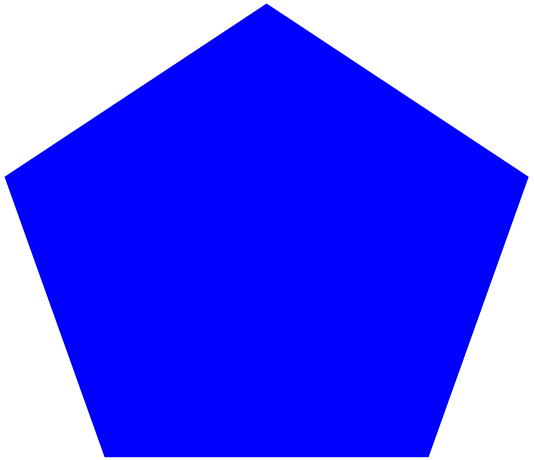


Round Shapes?



Legacy LibGDX: ShapeRenderer

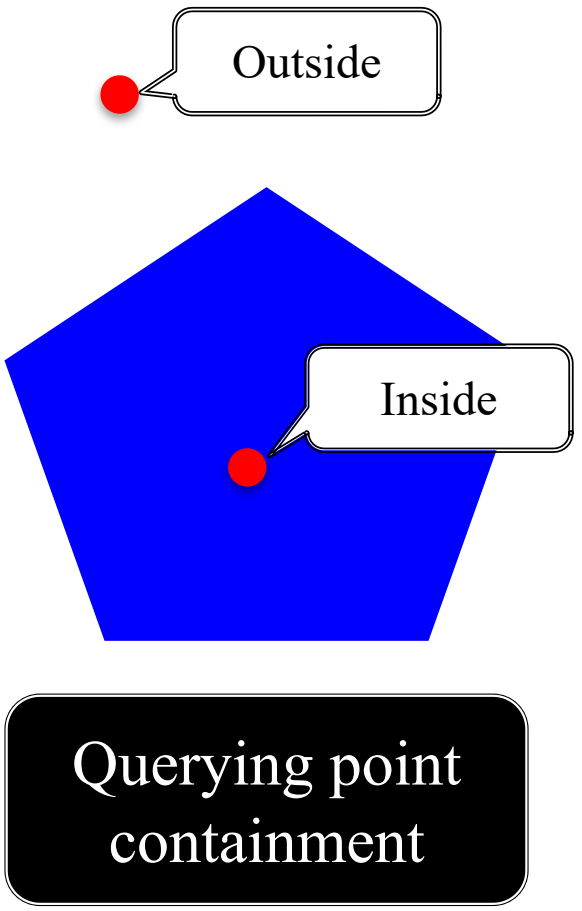
- Tool to draw triangles
  - Specify a general shape
  - Makes the triangles for you
- Works like a SpriteBatch
  - Has a begin/end
  - Several draw commands
- Hard to mix with SpriteBatch
  - Requires a separate **pass**
  - End one before begin other
  - Made our code really messy



```
render.circle(200, 200, 100, 5);
```

The Problem with ShapeRenderer

- It is just a drawing tool
  - Only draws shapes on screen
  - Does not keep any **geometry**
- Shapes are very limited
  - Triangles, rectangles, circles
  - Often a lot of manual work
- Conflates two issues
  - Computational geometry
  - Shape rendering
- We need a better solution



Computational Geometry

- Study of the **representation** of geometric shapes
  - Data structures to store complex shapes
  - Algorithms to generate complex shapes
- GDIAC tools allow us to do more than drawing
  - Query point containment
  - Intersect two shapes with each other
  - Break up a shape into triangles
  - Control Bezier splines
- Found in package `edu.cornell.gdiac.math`

Computational Geometry

- Study of the **representation** of geometric shapes
  - Data structures to store complex shapes
  - Algorithms to generate complex shapes
- GDI
  - Qu
  - Int
  - Break up a shape into triangles
  - Control Bezier splines

GDIAC SpriteBatch makes ShaperRenderer redundant

- Found in package `edu.cornell.gdiac.math`

The Primary Classes

Poly2

- Represents a **solid** shape
- Built from **triangles**
- `PolyFactory` for basic shapes
- Draw with `fill` method
  - Use Affine2 to position
  - Can be colored or textured

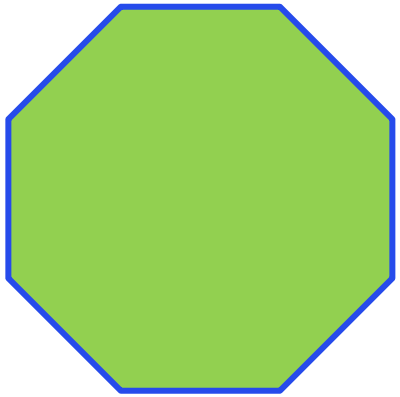
Path2

- Represents a shape **outline**
- Built from **line segments**
- `PathFactory` for basic shapes
- Draw with `outline` method
  - Use Affine2 to position
  - Can be colored or textured

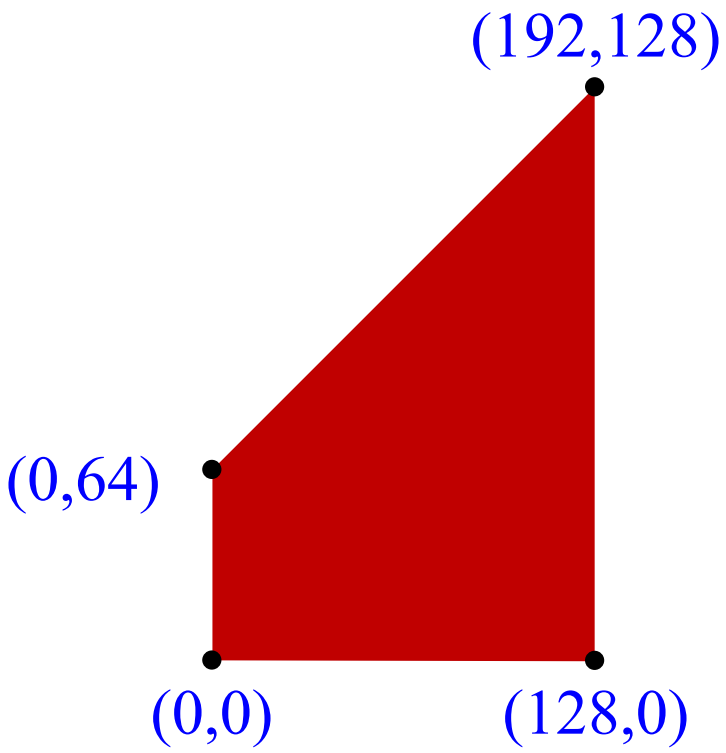
These are also useful for physics!

Example Using Factories

```
PathFactory pathTool = new PathFactory();
PolyFactory polyTool = new PolyFactory();

Poly2 interior = polyTool.makeNgon(200, 200, 100, 8);
Path2 border = pathTool.makeNgon(200, 200, 100, 8);

batch.begin(camera);

batch.setColor(Color.GREEN);
batch.fill(interior, transform);

batch.setColor(Color.BLUE);
batch.outline(border, transform);

batch.end();
```



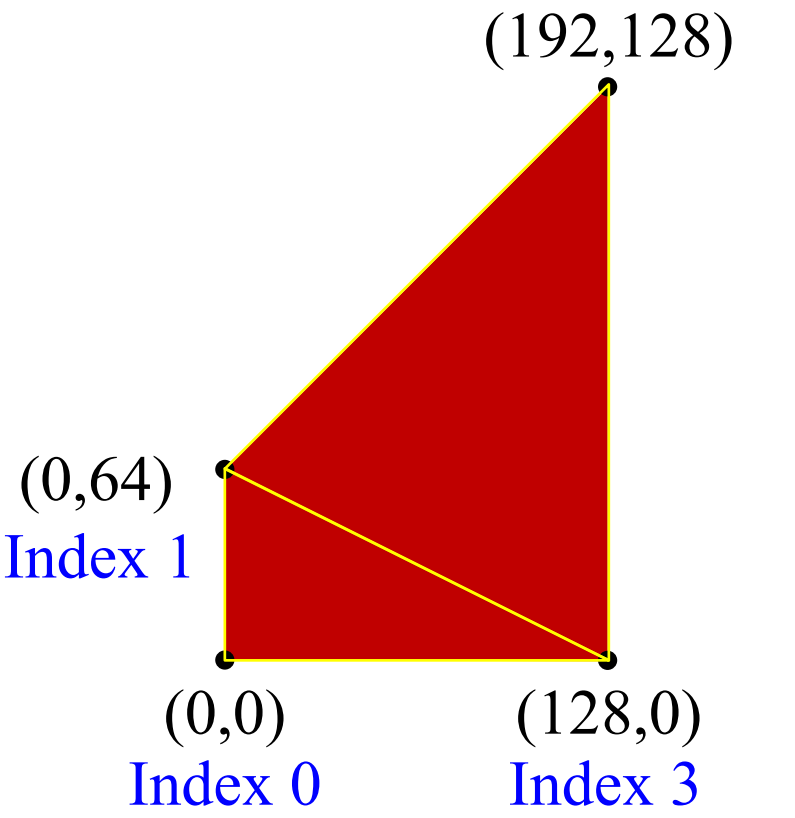
Understanding Poly2




```
verts = {0,0,0,64,192,128,128,0}
```

- Built up from **vertices**
  - Coordinates in 2d space
  - Stored as an array of floats
  - Transform with Affine2
- Must convert into triangles
  - Each vertex has an index
  - Given by position in array
  - Create array of indices
- `Path2` is much simpler
  - Only vertices in array
  - Connects adjacent vertices

Understanding Poly2

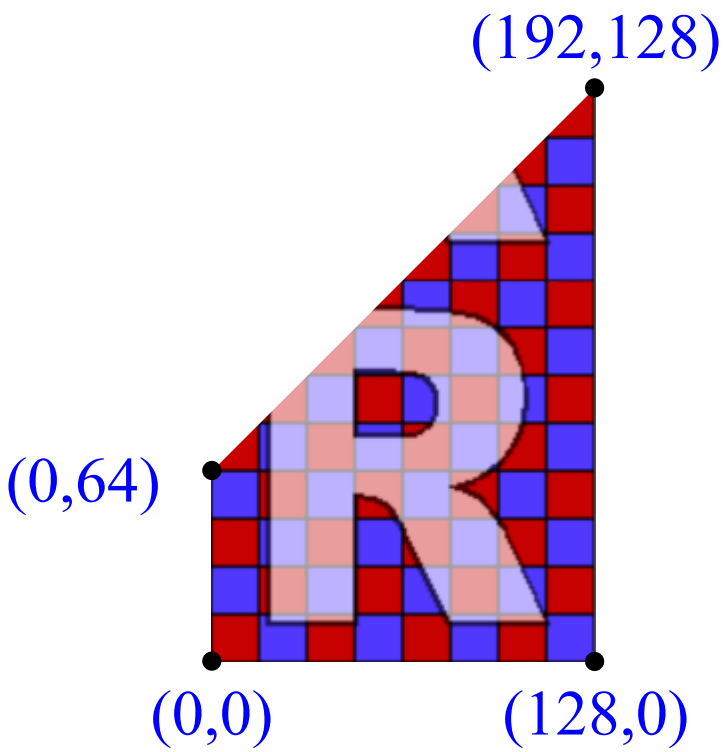


verts = {0,0,0,64,192,128,128,0}

tris = {0,3,1,3,2,1}

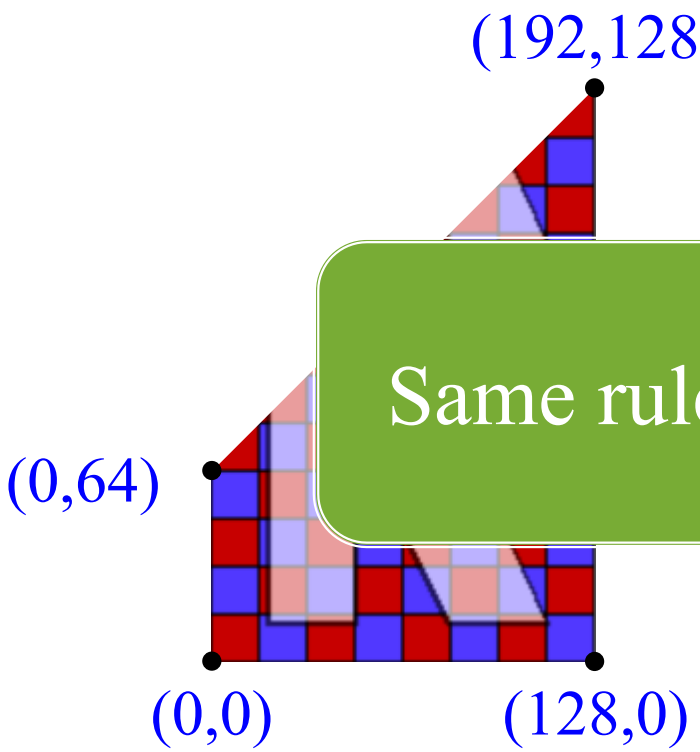
- Built up from **vertices**
  - Coordinates in 2d space
  - Stored as an array of floats
  - Transform with Affine2
- Must convert into triangles
  - Each vertex has an index
  - Given by position in array
  - Create array of indices
- `Path2` is much simpler
  - Only vertices in array
  - Connects adjacent vertices

Poly2 and Textures



- Poly2 can be **textured**
  - Fill is an image, not a color
  - Solid colors use null texture
- Works like a **cookie cutter**
  - Treat as object coordinates
  - Vertices are pixel locations
  - Uses image inside of shape
- What if goes out of bounds?
  - Example texture is 128x128
  - Depends on texture settings
  - Options: **repeat** or **clamp**

Poly2 and Textures

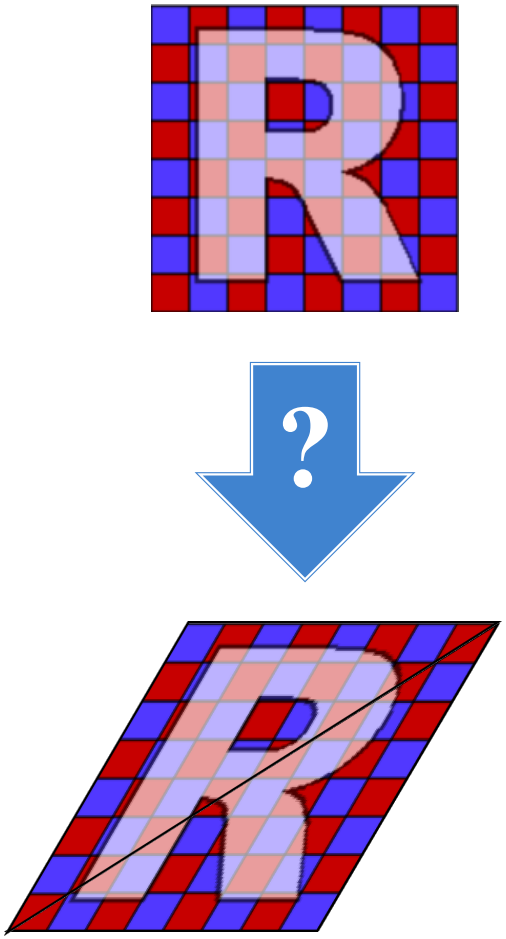


Same rules apply to Path2

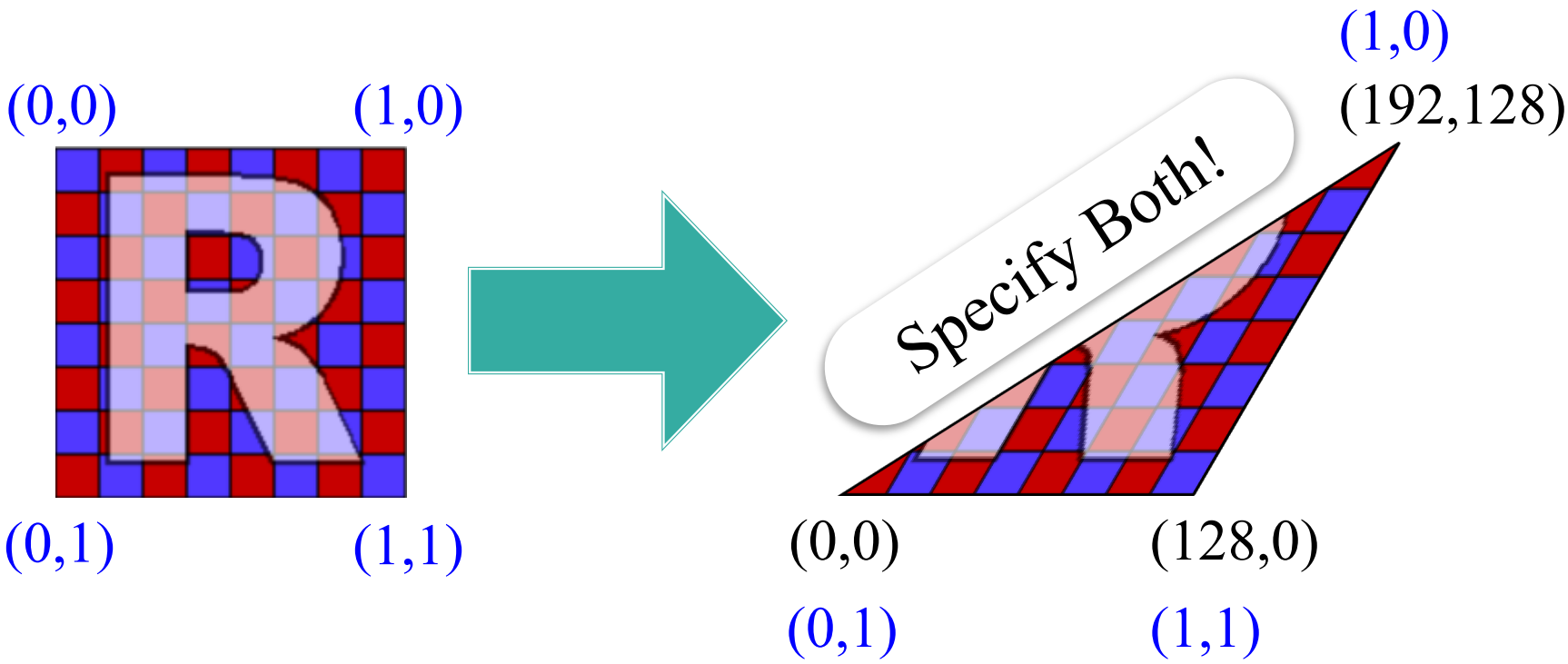
- Poly2 can be **textured**
  - Fill is an image, not a color
  - Solid colors use null texture
- Works like a **cookie cutter**
  - t coordinates
  - xel locations
  - side of shape
- What if goes out of bounds?
  - Example texture is 128x128
  - Depends on texture settings
  - Options: **repeat** or **clamp**

Limitation of Poly2

- Texturing is by **cookie cutter**
  - Stretching shape changes area
  - Does **not** stretch the image
  - Also depends on image **size**
- Want more design flexibility
  - Want to "pin" image to shape
  - Image moves with vertices
- **Idea**: Two sets of vertices
  - One to represent the Poly2
  - Another for the image pixels
  - Must support texture swapping

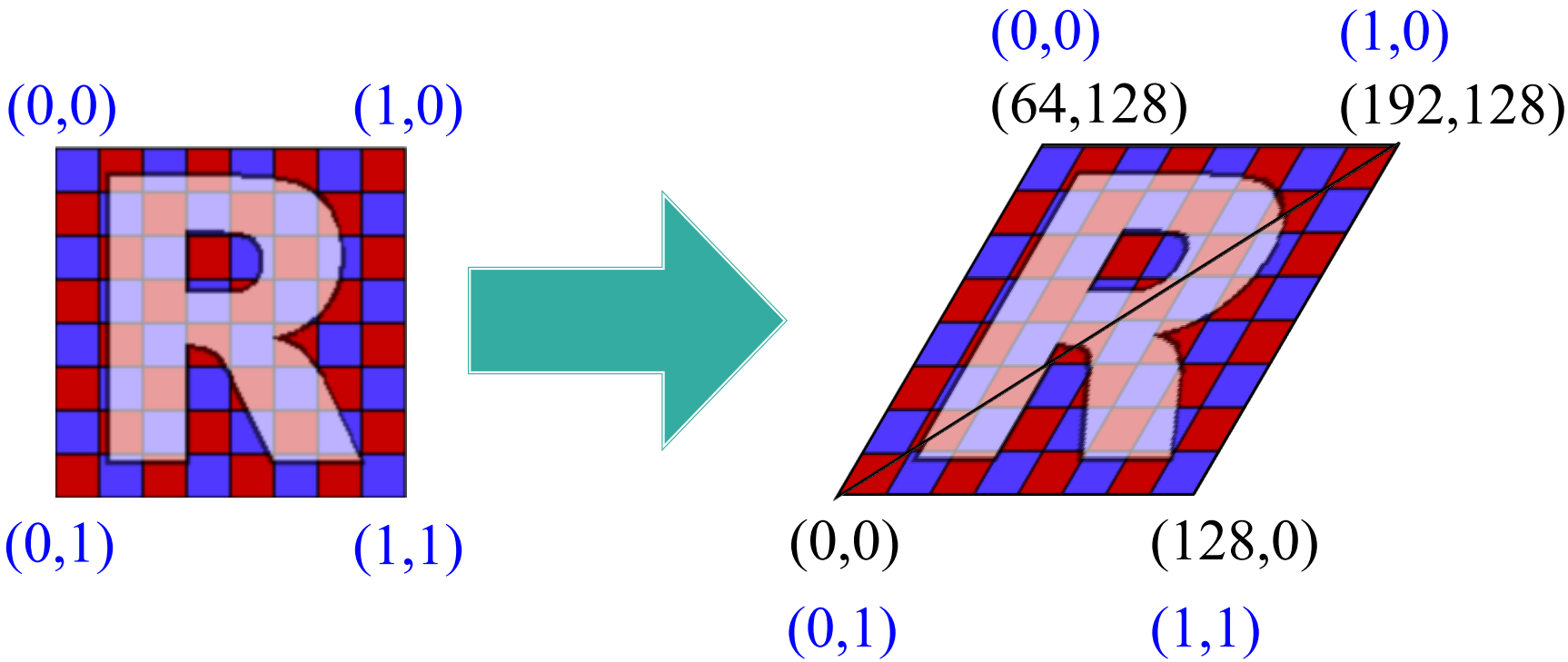


Texture Coordinates are Percentages



Texture Coordinates
(even if not square)

Texture Coordinates are Percentages

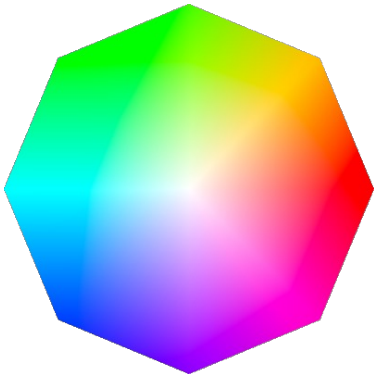


Texture Coordinates
(even if not square)

Triangle Coordinates
(more than one triangle)

GDIAC Support: SpriteMesh

- Very similar to `Poly2`
  - Specify individual vertices
  - Specify triangles as indices
  - Constructor takes a `Poly2`
- But for each vertex can add
  - Individual color
  - Texture coordinates
  - Gradient coordinates (?)
- Used heavily in **Lab 4**
  - Build Poly2 for box2d
  - Then define a `SpriteMesh`

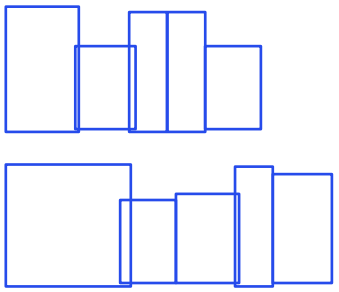


SpriteBatch Revisited

- `SpriteMesh` objects are **composable**
  - Each mesh has two arrays: vertices, indices
  - Concatenating vertices makes a new mesh
  - Indices are added and concatentated
- `SpriteBatch` makes one large `SpriteMesh`
  - Each draw command adds to the mesh
  - The final `end()` draws the mesh to the screen
  - **Exception:** Also draws if mesh gets too large
- Very efficient for sending to graphics card

SpriteBatch and Text

- Each `Font` has a texture
  - Image with letters pre-drawn
  - Reason you specify font size
- `TextLayout` makes a **mesh**
  - Positions from font metrics
  - Includes *kerning*, *alignments*
  - Vertices include texture cords
- This makes text **very fast**
  - Generating vertices is quick
  - Actual font cached in texture

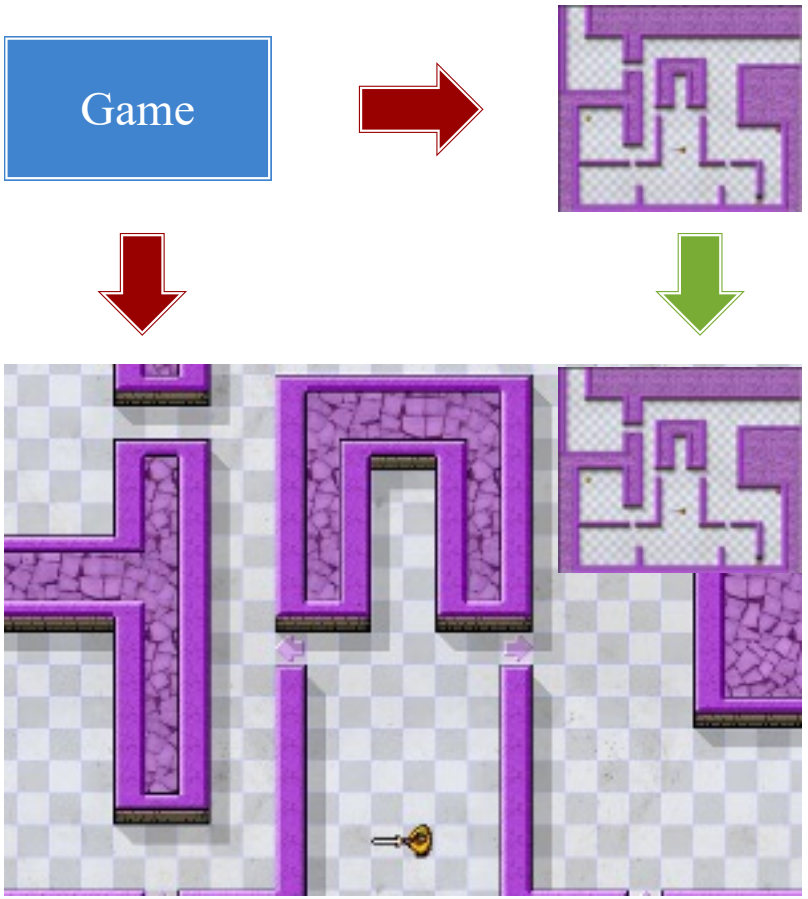


SpriteBatch and Text

- Each `Font` has a texture
  - Image with letters pre-drawn
  - Reason you specify font size
- `TextLayout` makes a **mesh**
  - Positions from font metrics
  - Includes *kerning*, *alignments*
  - Vertices include texture cords
- This makes text **very fast**
  - Generating vertices is quick
  - Actual font cached in texture

Offscreen Rendering

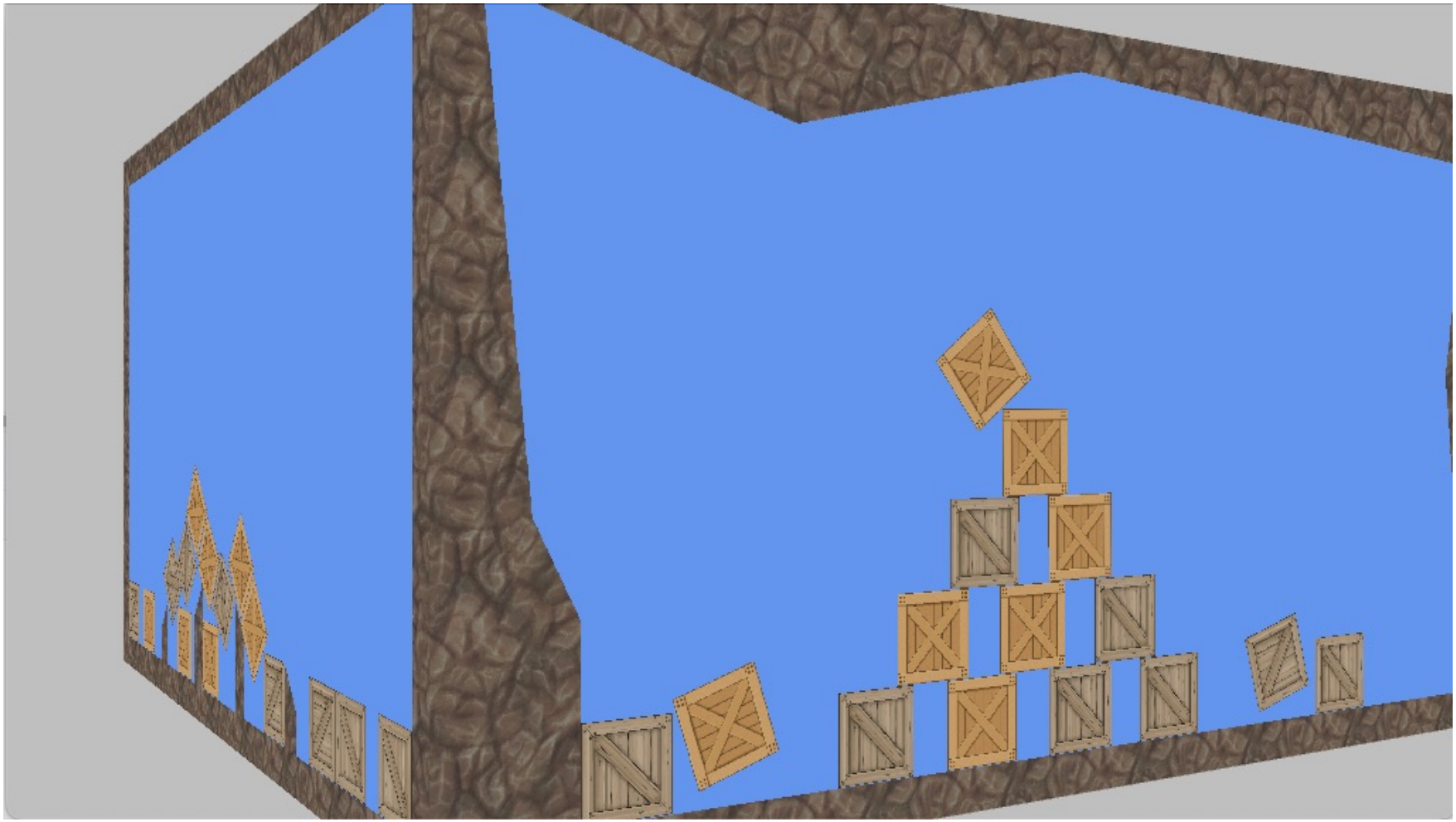
- Can create *dynamic* textures
  - Draw an image to a texture
  - Draw that texture to screen
  - Called *offscreen rendering*
- Applications
  - Mini-maps
  - Scene Transitions
  - Light & Shadow
- Made easy in GDIAC
  - Class `RenderTarget`
  - Uses begin/end pattern



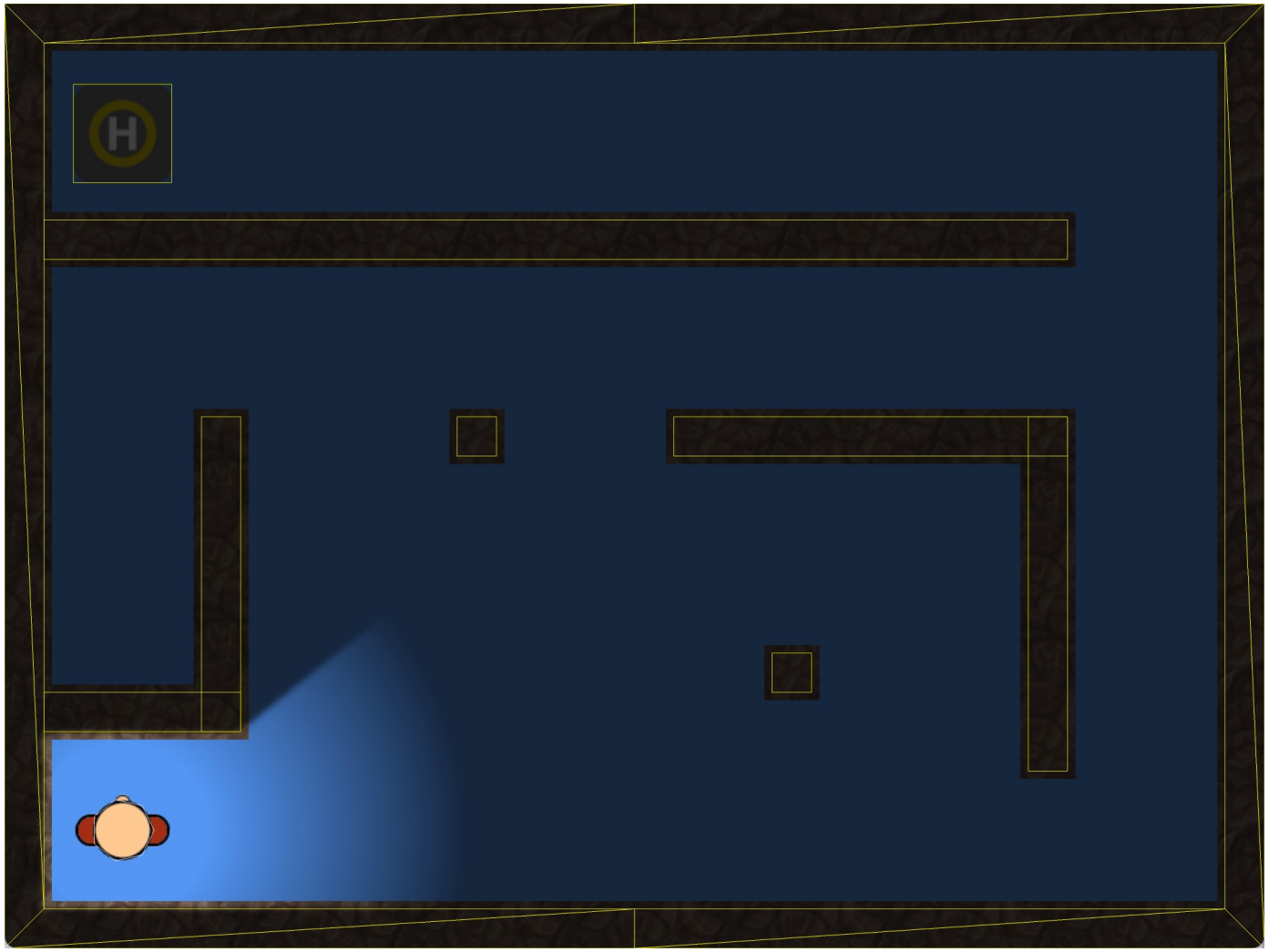
Using RenderTarget

```
// Set the resolution of the texture
RenderTarget target = new RenderTarget(width, height)

target.begin();

// All drawing here is done offscreen
batch.begin(camera);  // Camera must match width/height
...
batch.end();

target.end();

Texture texture = target.getTexture();

// Now use texture to draw to screen
```

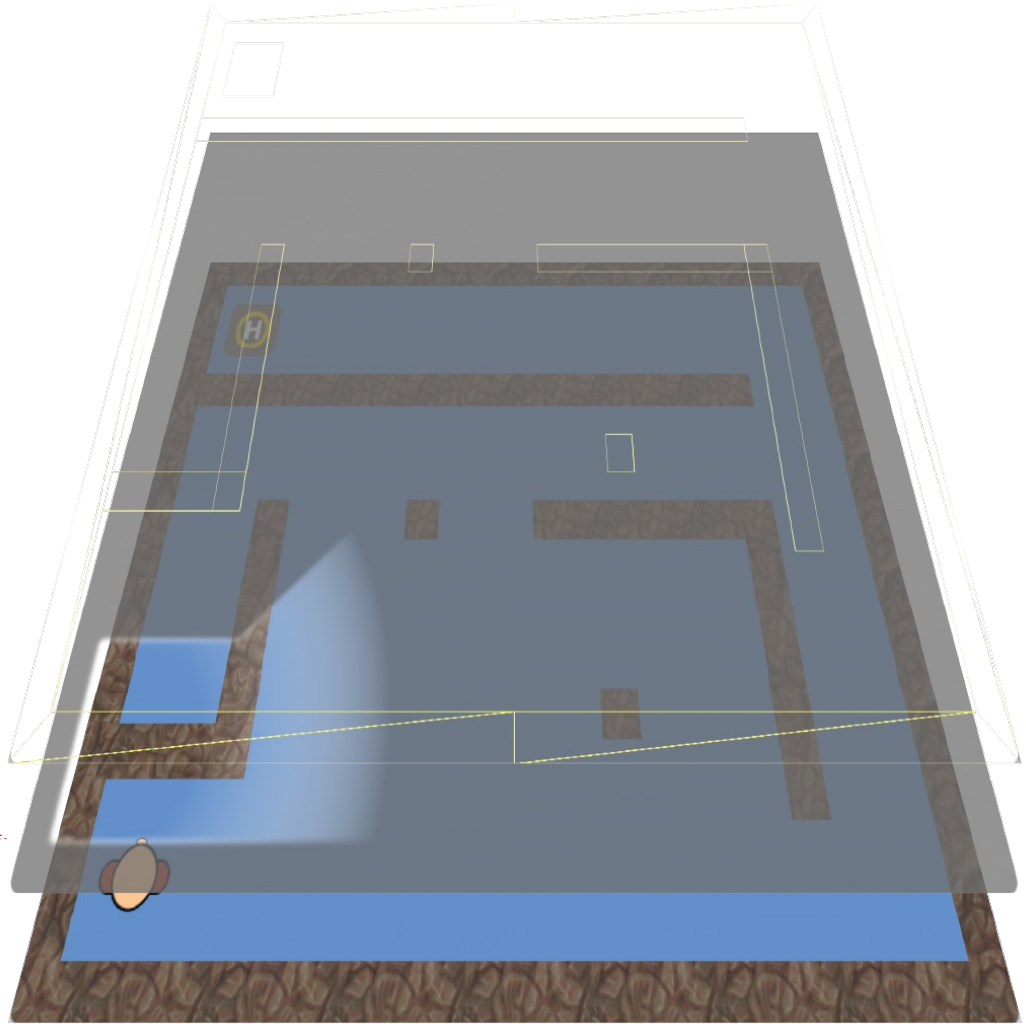
Example: Cube Transitions



Example: box2d Lights



Example: box2d Lights



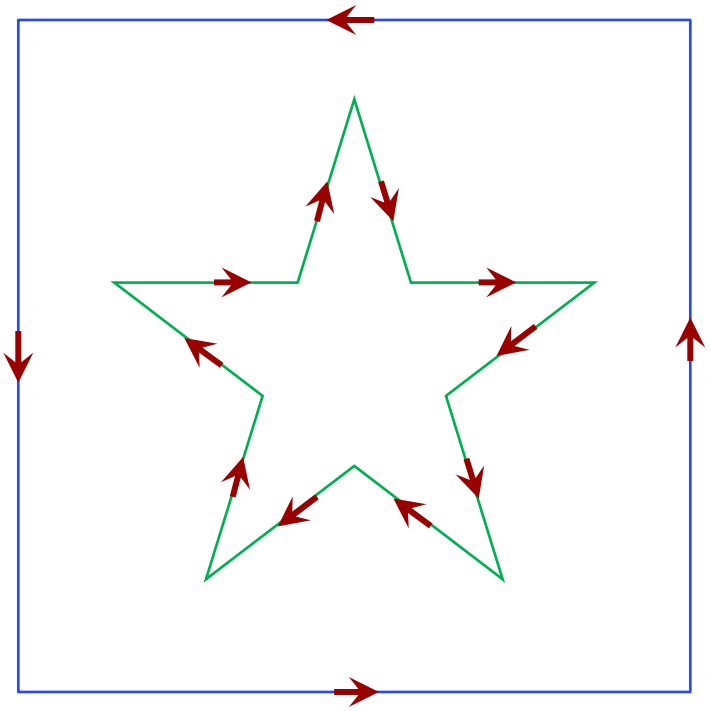
Drawn to texture

Drawn to screen

Drawn to screen

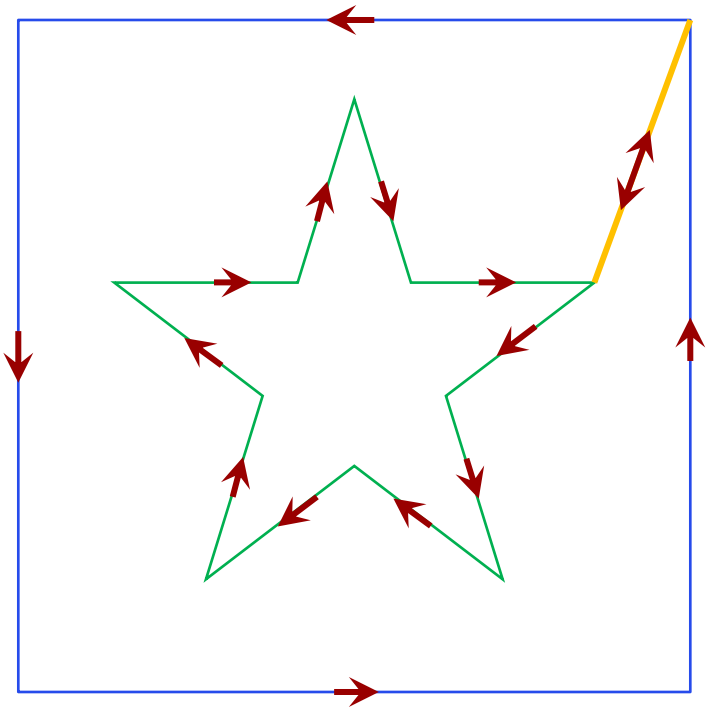
Additional Techniques: Triangulation

- **Goal:** Convert `Path2` to `Poly2`
  - Fill insides with triangles
  - Avoid adding new vertices
- Requirements for `Path2`
  - Lines cannot cross
  - Must be counter-clockwise
  - Clockwise paths are "holes"
- Class `PolyTriangulator`
  - Uses **ear-clipping** algorithm
  - Simple and fast enough
  - But triangles are not optimal



Additional Techniques: Triangulation

- **Goal:** Convert `Path2` to `Poly2`
  - Fill insides with triangles
  - Avoid adding new vertices
- Requirements for `Path2`
  - Lines cannot cross
  - Must be counter-clockwise
  - Clockwise paths are "holes"
- Class `PolyTriangulator`
  - Uses **ear-clipping** algorithm
  - Simple and fast enough
  - But triangles are not optimal



Additional Techniques: Triangulation

- **Goal:** Convert Path2 to Poly2
  - Fill insides with triangles
  - Avoid adding new vertices
- Requirements for Path2
  - Lines cannot cross
  - Must be counter-clockwise
  - Clockwise paths are "holes"
- Class PolyTriangulator
  - Uses **ear-clipping** algorithm
  - Simple and fast enough
  - But triangles are not optimal

Additional Techniques: Triangulation

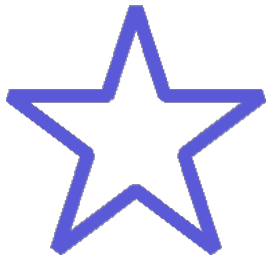
- **Goal:** Convert Path2 to Poly2
  - Fill insides with triangles
  - Avoid adding new vertices
- Requirements for Path2
  - Lines cannot cross
  - Must be counter-clockwise
  - Clockwise paths are "holes"
- Class PolyTriangulator
  - [illegible]
  - [illegible]
  - [illegible]

Alternative algorithms are a student project

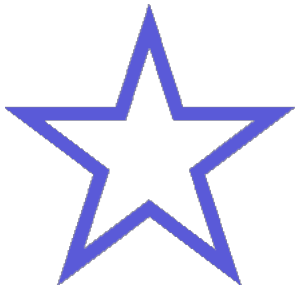
Additional Techniques: Extrusion

- Often don't draw `Path2` objs
  - Lines only one pixel wide
  - Too thin for hi-res monitors
  - Only useful for debugging
- Want `Path2` to have *width*
- **Idea:** convert to a `Poly2`
  - But not triangulation!
  - **Extrude** volume along path
- Some choices to make
  - Shape of ends
  - Shape of corners

SQUARE



MITRE



ROUND

Additional Techniques: Extrusion

- Often don't draw `Path2` objs
  - Lines only one pixel wide
  - Too thin for hi-res monitors
  - Only useful for debugging
- Want `Path2` to have *width*
- **Idea:** convert
  - But not trian
  - **Extrude** volume along path
- Some choices to make
  - Shape of ends
  - Shape of corners

Class PathExtruder

Using the GDIAC Tools

PolyTriangulator

```
Poly2 triangulate(Path2 path) {

    PolyTriangulator triang;
    Poly2 result;

    triang = new PolyTriangulator();
    triang.set(path);

    // Add any holes here

    triang.calculate();
    result = triang.getPolygon();

    return result;
}
```

PathExtruder

```
Poly2 extrude(Path2 path) {

    PathExtruder extruder;
    Poly2 result;

    extruder = new PathExtruder();
    extruder.set(path);

    // Set cap, joints here

    extruder.calculate(width);
    result = extruder.getPolygon();

    return result;
}
```

Using the GDIAC Tools

PolyTriangulator

```
Poly2 triangulate(Path2 path) {

    PolyTriangulator triang;
    Poly2 result;

    triang = new PolyT
    triang.set(path);

    // Add any holes he

    triang.calculate();
    result = triang.getPolygon();

    return result;
}
```

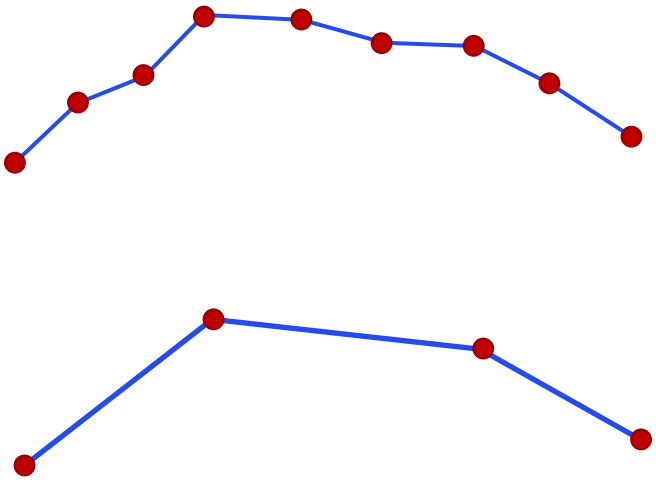
PathExtruder

```
Poly2 extrude(Path2 path) {

    PathExtruder extruder;
    Poly2 result;

                          PathExtruder();
                        h);

                        s here

    extruder.calculate(width);
    result = extruder.getPolygon();

    return result;
}
```

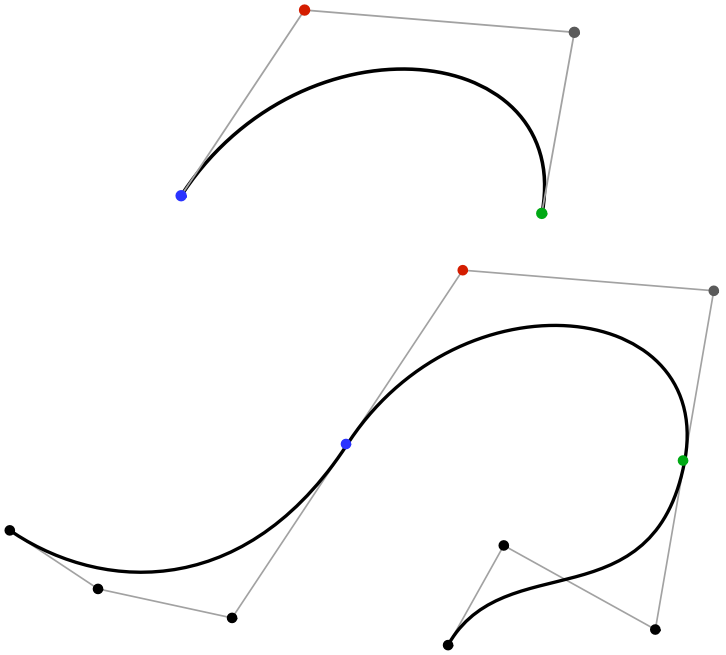
Very similar structure

Other GDIAC Features

Path Smoothing



Splines



Other GDIAC Features

Path Smoothing

Splines

Summary

- GDIAC extensions have many new features
  - Leverages strength of OpenGLES 3.0
  - But still backwards compatible with LibGDX
- Tools for **computation geometry**
  - Factories for `Poly2` and `Path2` objects
  - Triangulation and extrusion
  - Spline support for curves
- Tools for **texture manipulation**
  - `SpriteMesh` is an easy extension of `Poly2`
  - `RenderTarget` makes offscreen rendering easy