# Lecture 24
# Performance Tricks

# Warning...

Graphics performance is a moving target

Focus on:

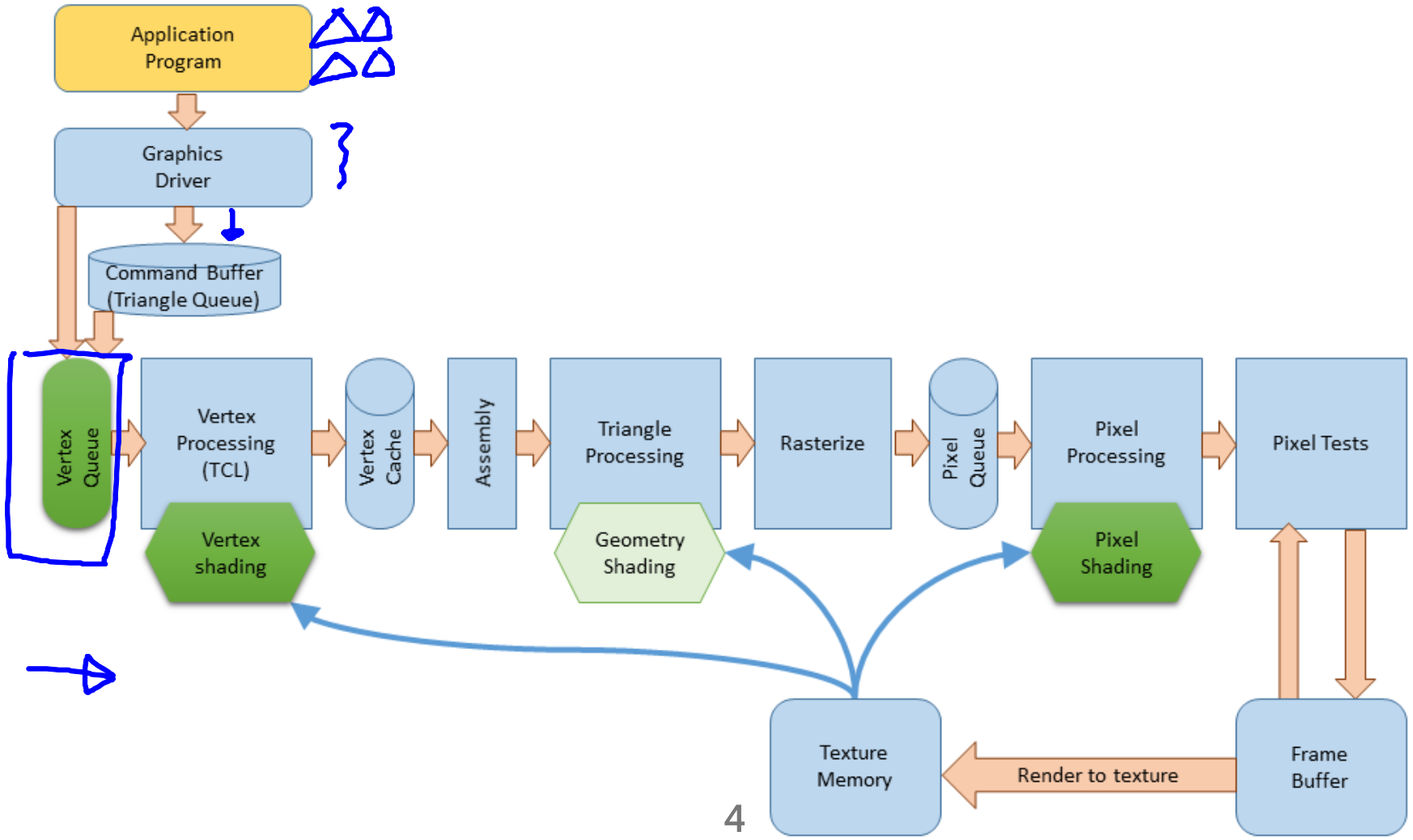
- methods that are generally useful (for other reasons)
- methods that are really important in 2022
- methods that are useful for your project

- methods that help us understand the pipeline

# Where are the bottlenecks?

- limited memory (space/performance) on textures
  - texture sharing
- want complex light setups
  - use environment maps for lighting
- keep the number of objects small
  - matrix pallete approaches (skinning)
- change shape without changing vertices
  - deformation-based approaches (skinning, FFD, cages, ...)
- shading fragments we never see
  - deferred sharing

# The Pipeline - where can things go wrong?



# How much work do we do?

- Process the triangle
- Rasterize
- Process Fragments
  - multiple texture lookups
  - texturing computations
  - shadow maps, environment maps, ...
  - potentially many lights, ...

And then...

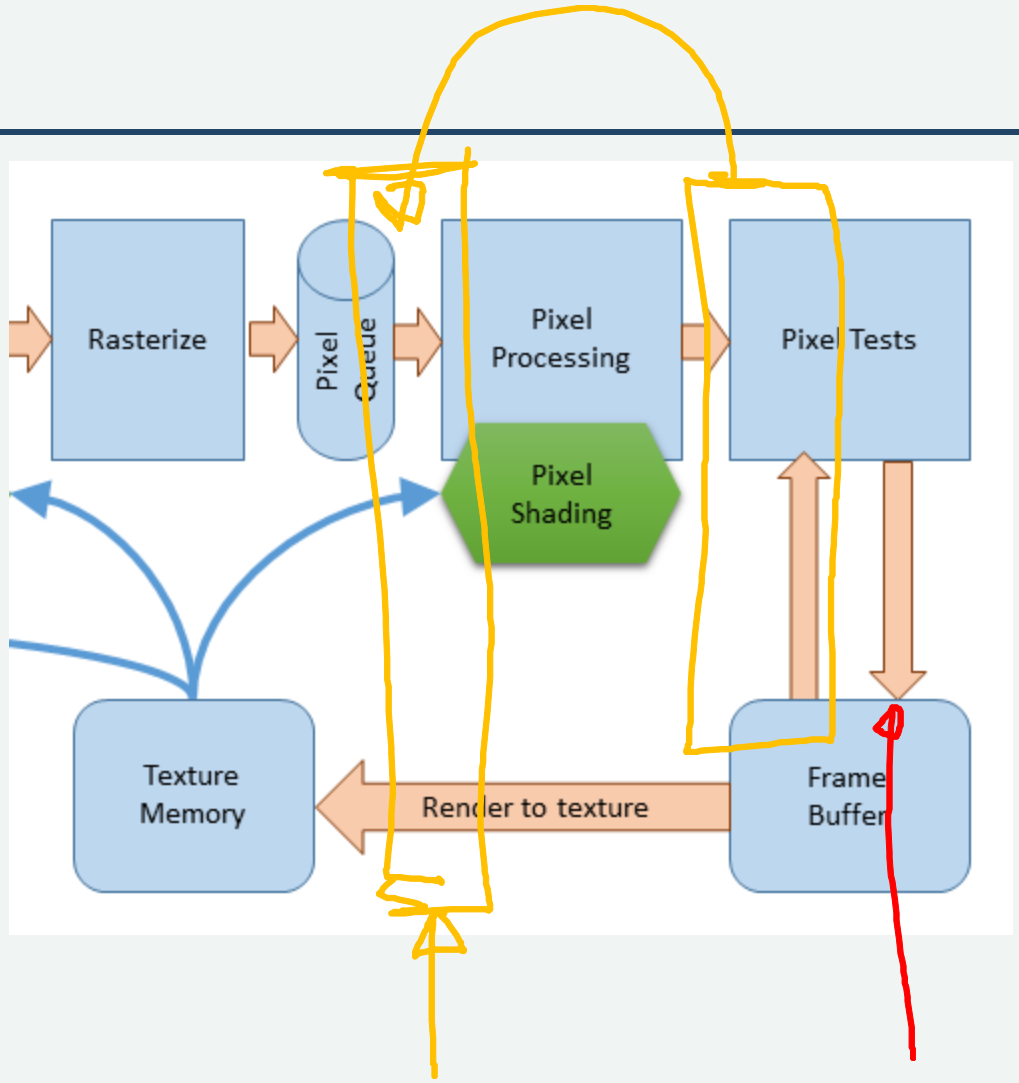
maybe it fails the Z-test

# Idea 1: Early Z

If the fragment shader doesn't change the Z coordinate...

Do the Z-Test **before** pixel processing

- this seperates pixel read from pixel write
- can create timing issues



# Idea 2: Two Passes

1. Draw with a really simple shader (just write Z)
2. Draw with the early Z-test (do Z equals)
    - in theory, only one fragment passes the test

Good:

- shade each pixel once
- no sorting required

Bad:

- two passes
- still need early Z cutoff

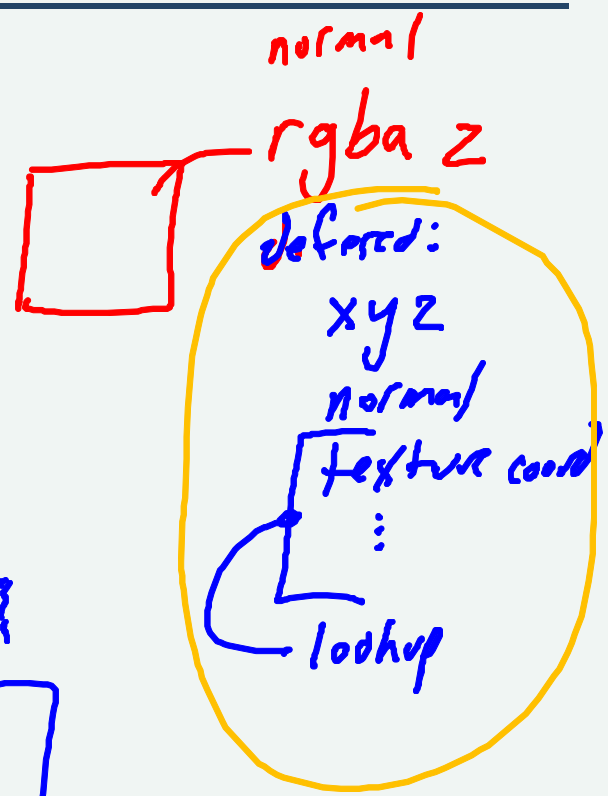
# Idea 3: Defered Rendering

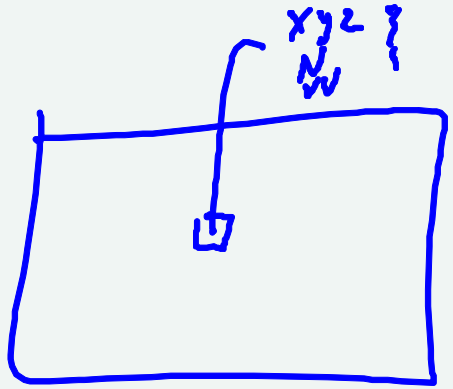
This is a big deal in modern games

This isn't something you do yourself in THREE

# Deferred Shading/Rendering

1. Don't shade the fragments when we draw them
2. Store information needed for shading in a **Geometry Buffer**
3. Make a second pass where colors are computed
    - possible multiple passes
    - each pixel is considered once
    - one big polygon that covers screen



# A Note on Terminology

Deferred **shading**

Deferred **rendering**

Deferred **lighting**

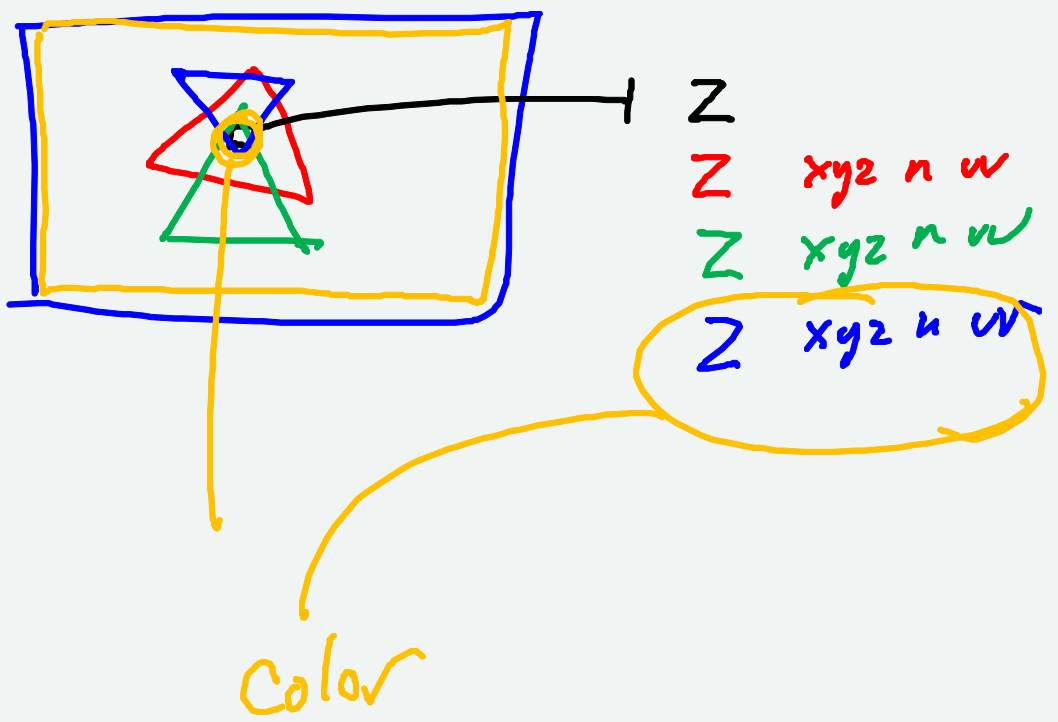
Different sources use these terms differently

The basic ideas are the same

Arguments as to which details go with different terms

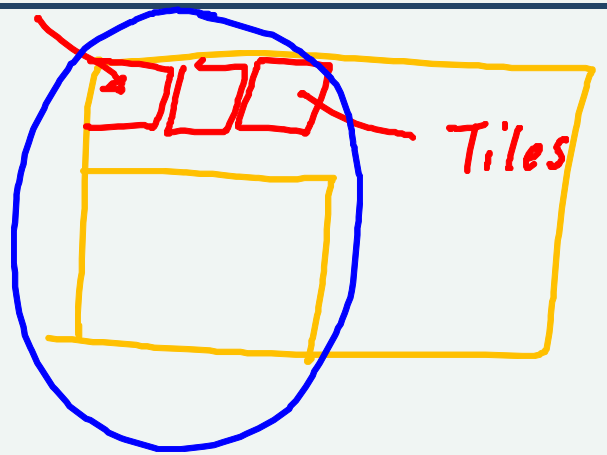
# Simple Deferred Shading Example

1. Draw all object w/Z-buffer
    - store in G-Buffer
2. Draw one big polygon (screen)
    - shade using G-Buffer



# Extensions

1. Loop over light sources (light sums in pixels)
2. Light sources don't need to do entire screen
3. Break things into **tiles**
    - break screen into regions
    - list of triangles for each region (culling)
    - list of lights for each region
    - much smaller memory footprint




Tile-based defered rendering - very common in games

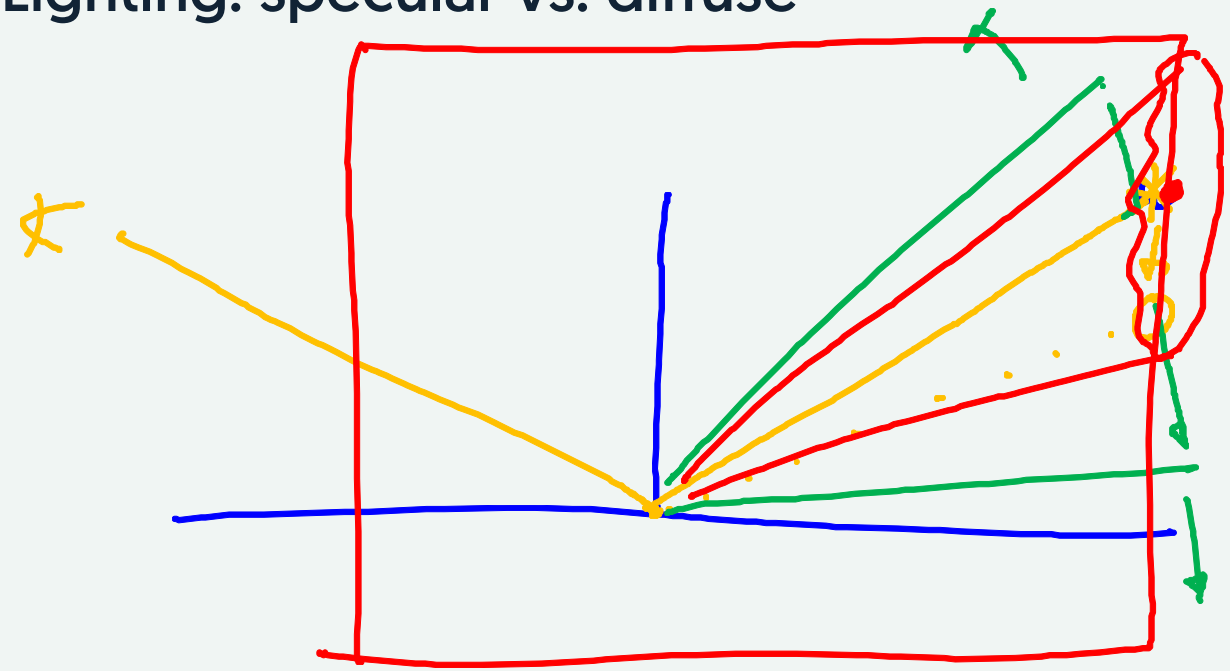
Mobile hardware is explicitly designed for TBDR

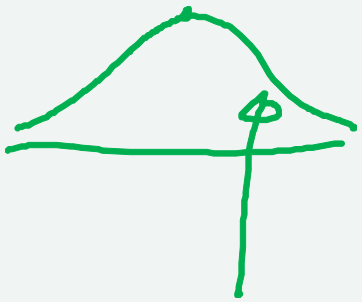
# Complex Lighting

- How do we have lots of lights?
- How do we have objects act as lights?

# Complex Lighting

Simple Lighting: specular vs. diffuse





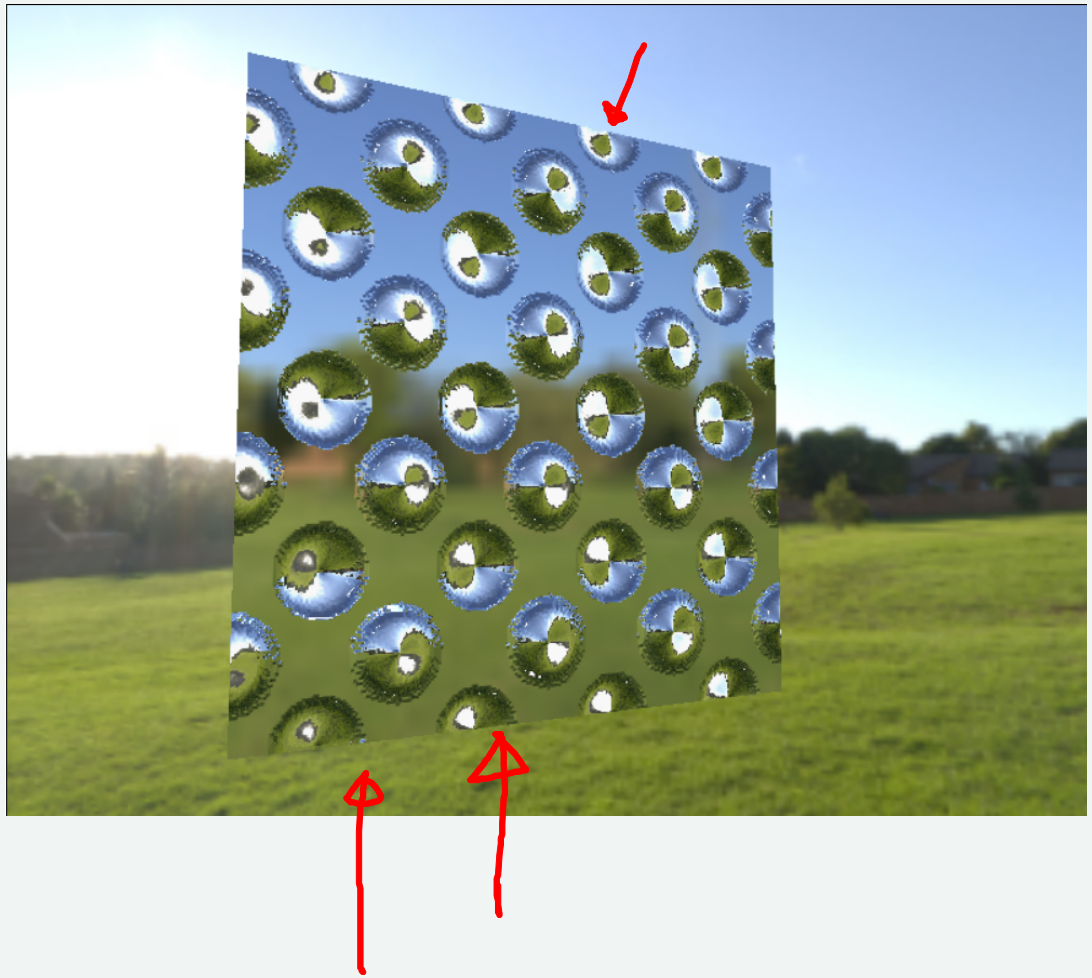
# Specular model

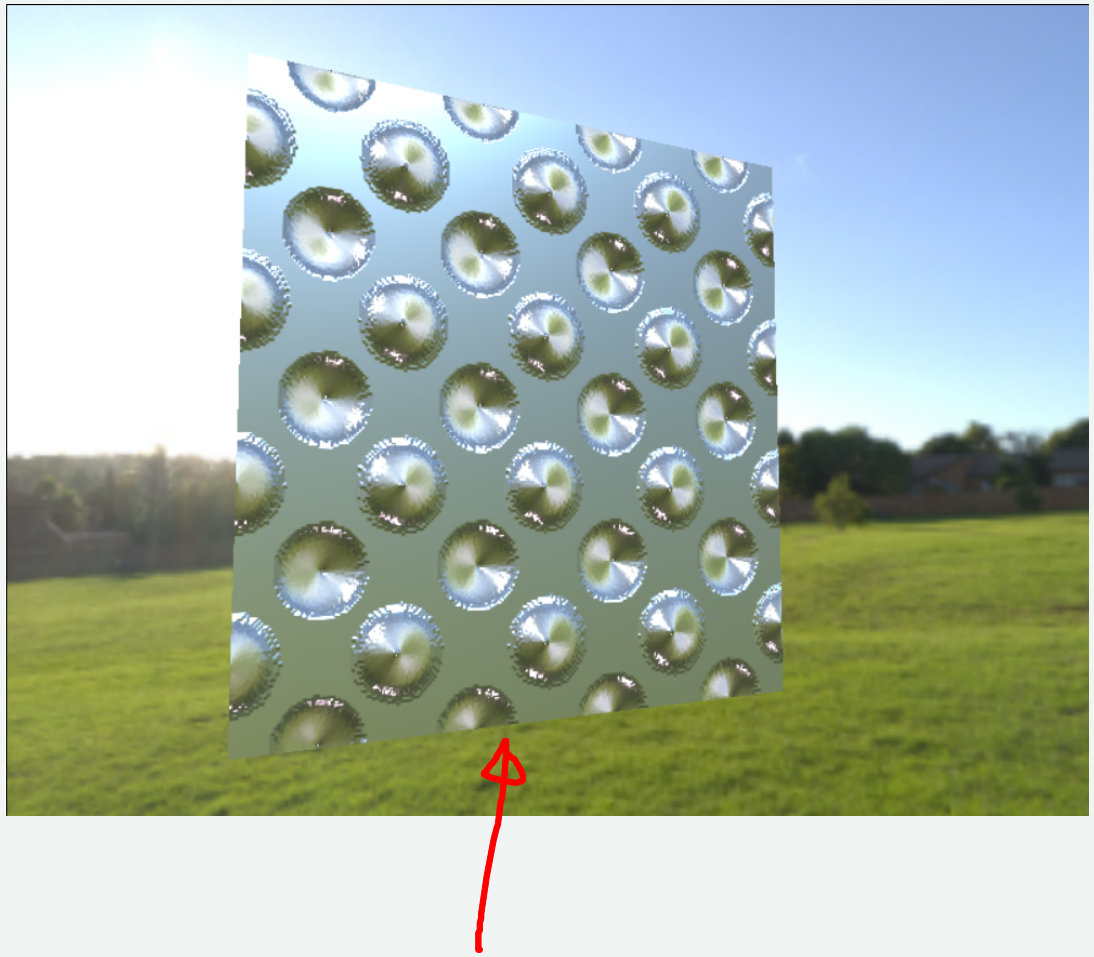
As "shininess" goes to zero, becomes (more like) diffuse

# Using with an environment map

As "shininess" goes down (roughness goes up), filter area goes up

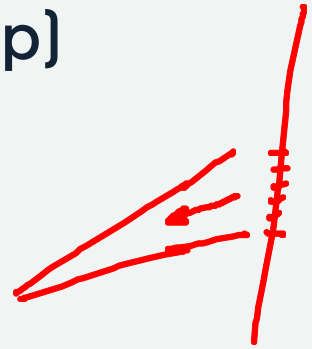
# Less "reflective"





# A Hack way to get fancy lighting

- Capture "main sources" in environment map
- Independent of number of lights
- Independent of complexity of lights
- Potentially Dynamic (with Dynamic Environment Map)
- Potentially capture real lighting
    - used for visual effects in movies



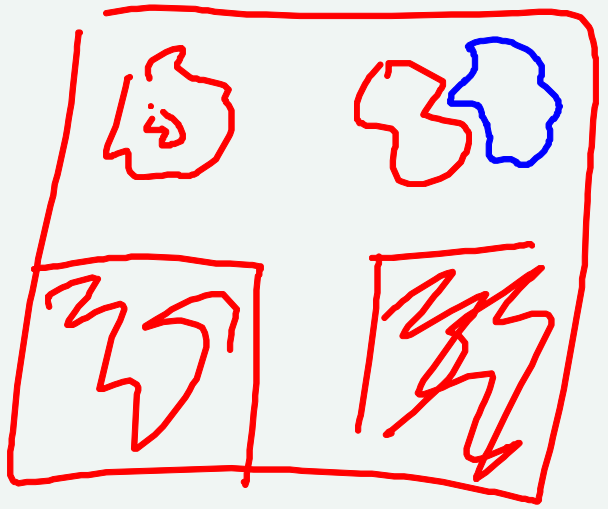
# Do We Need So Many Textures

Textures are expensive!

- need to load them
- need to process them
- need to store them
- need to set them up
- need to read from them (caching)

# Tricks to make it work better

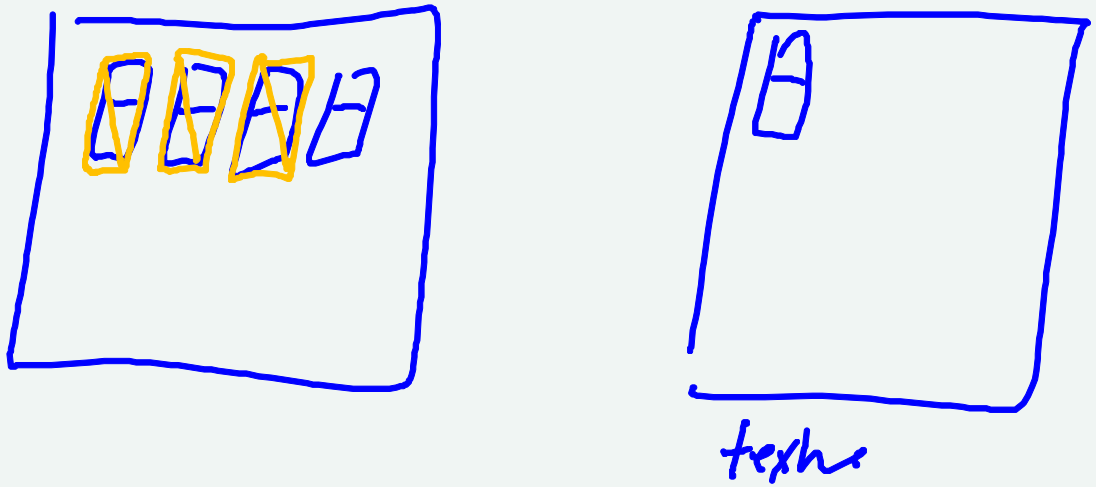
- reuse (load the texture once)
- put many textures in one image
  - Texture Atlas



# Why this can really help

One big object, one "material"

minimize **state changes**



# One big object?

What if we want to transform parts differently?

# Matrix Palette

1. Pass multiple matrices (an array of them)
2. Each vertex specifies which matrix it is part of
    - attribute

# Specifying which matrix

1. Give the number of the matrix
2. Give a vector of weights
    - allows for blending

This is really a setup for skinning...



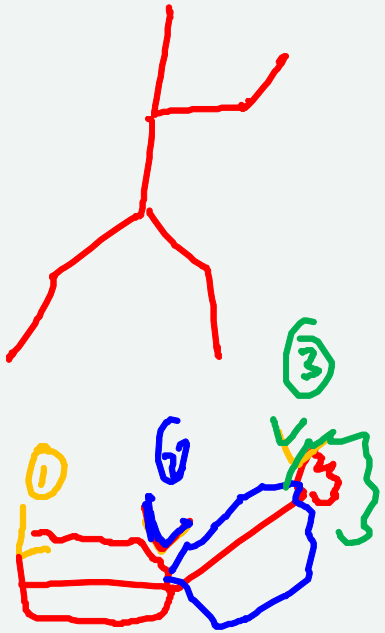
V4 .5 .5

# Skinning

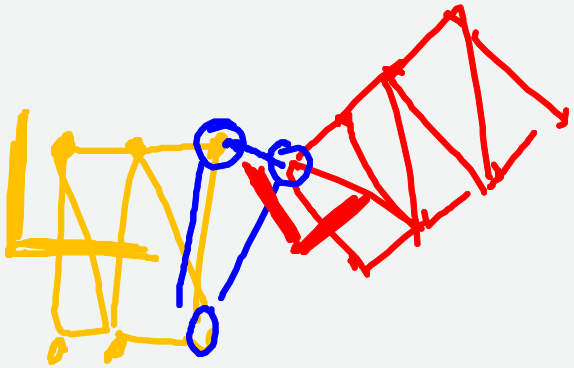
# Skeletons

Bones as coordinate systems

- how to specify movements?
  - inverse kinematics
  - motion capture
- how to draw the character

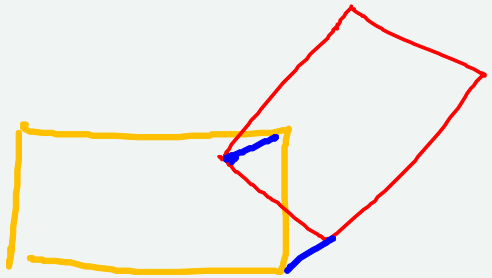


# Rigid pieces



# Skinning Concepts

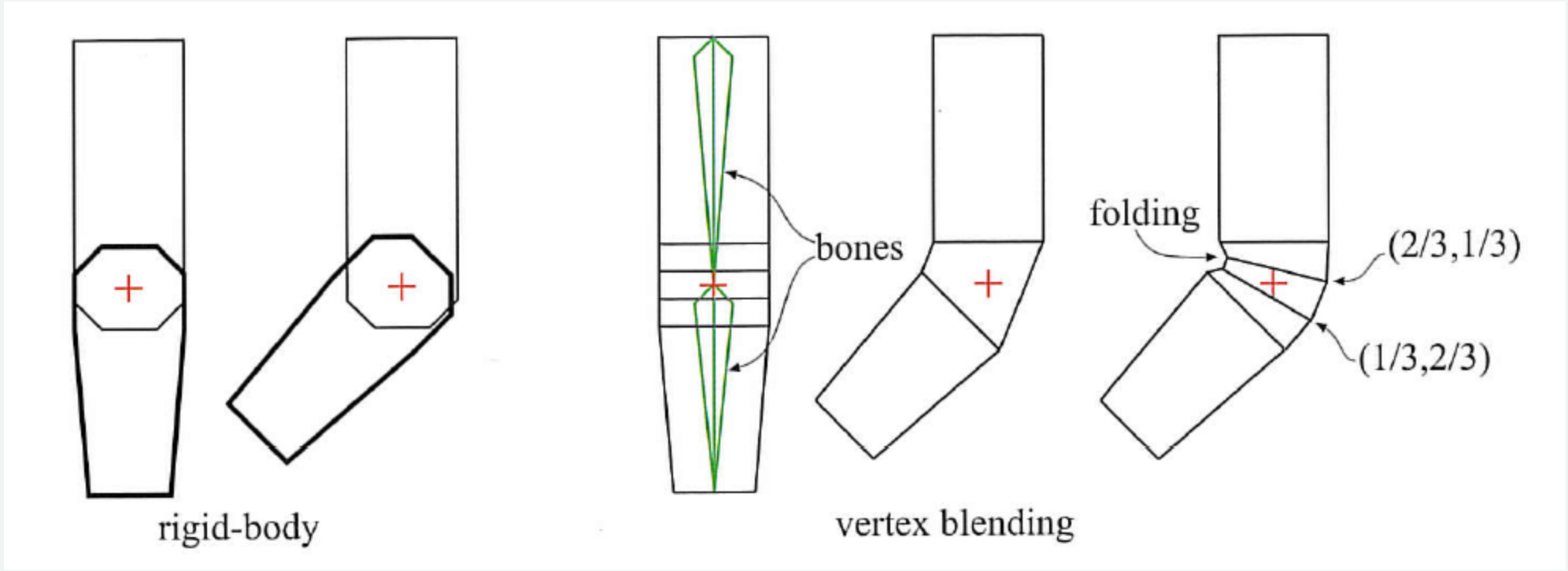
1. Bones (Transforms)
2. Binding pose
3. Attachment

# Skinning

## Linear Blend Skinning

- Each vertex in multiple coordinate systems (weighting)

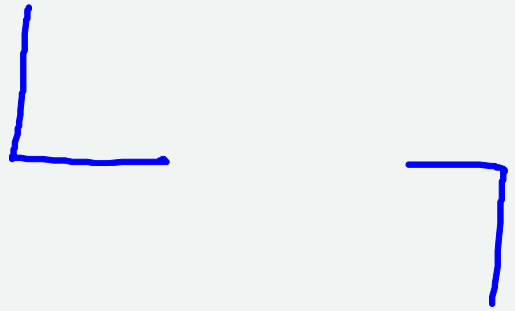

rigid-body
bones
vertex blending
folding
(2/3,1/3)
(1/3,2/3)

# Blend Matrices?

$$p_s = \alpha M_1 p_0 + \beta M_2 p_0$$

is the same as

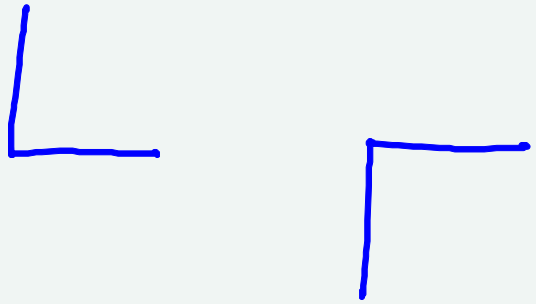
$$p_s = (\alpha M_0 + \beta M_2) p_0$$

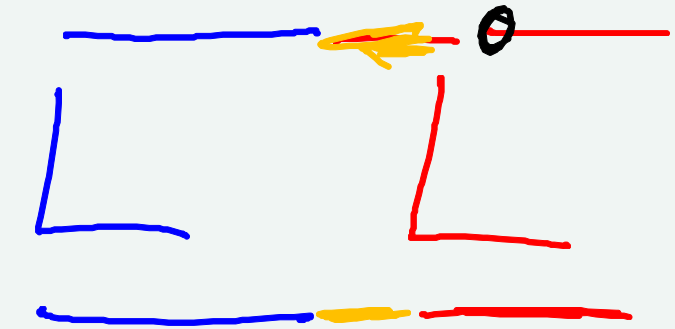


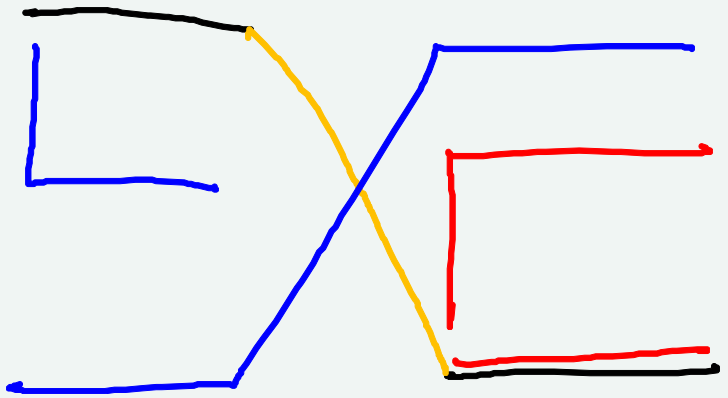
If $M_1$ and $M_2$ are rotations, scaling/adding doesn't give a rotation!

This leads to weird artifacts...

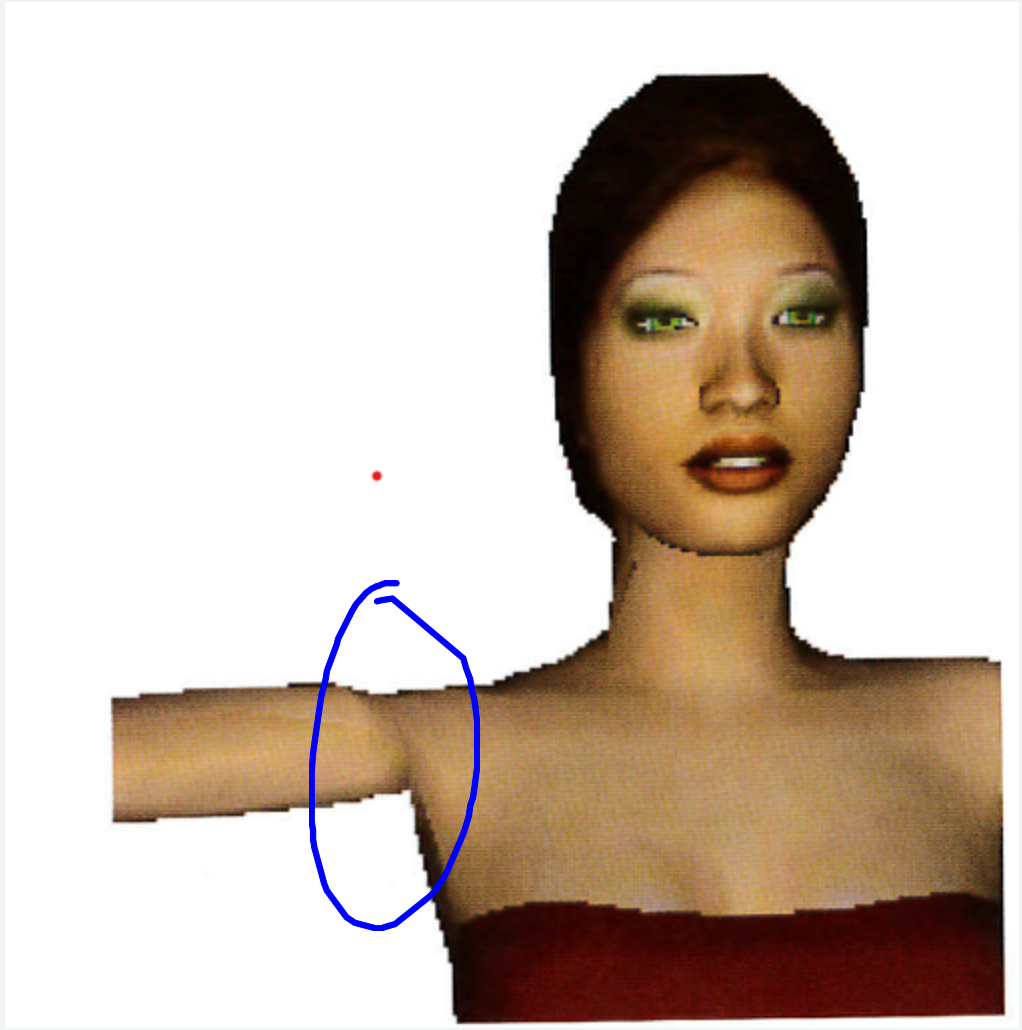
# Blend Artifacts

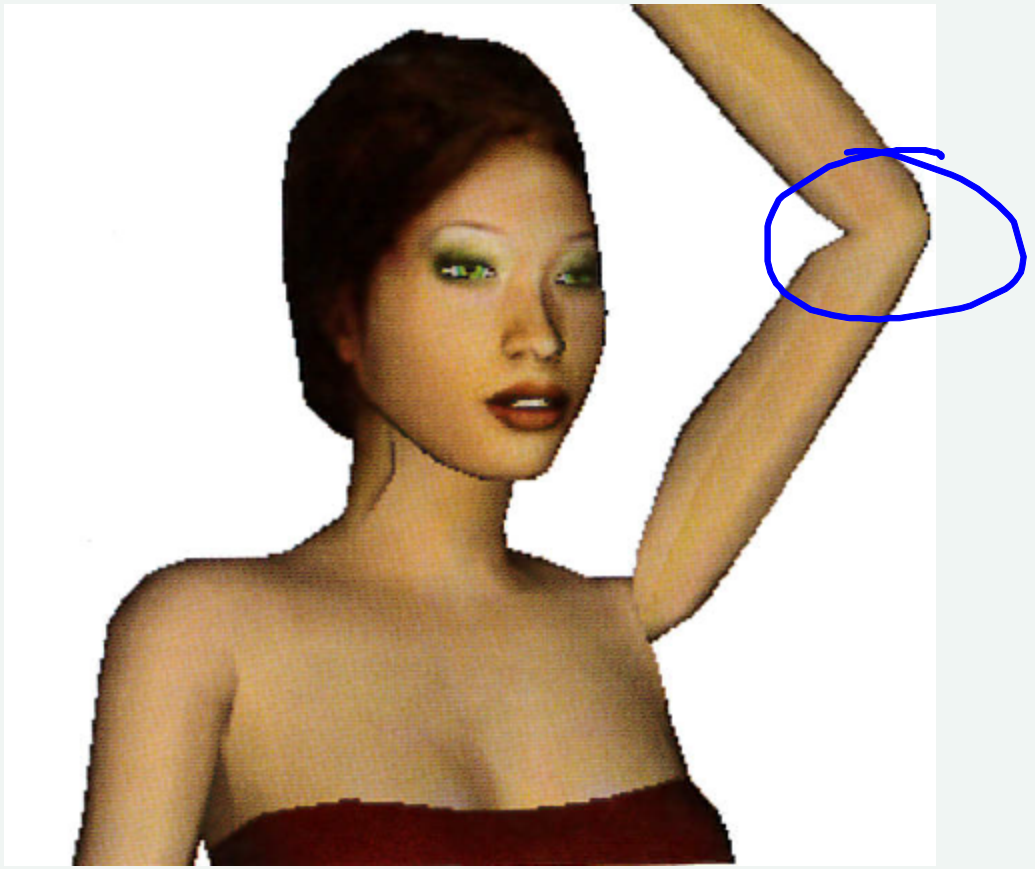






# Blend artifacts





# Linear Blend Skinning?

- Popular because it is simple
- Artists work around artifacts
- More complex alternatives exist - trade complexity for quality

# More Generally...

Make one shape

Deform it to other shapes

- easier to animate
- easier to model/control

# Animation by Deformation

Advantages:

1. performance (don't need to compute every vertex)
    - no need to send mesh to graphics hardware each frame
    - per-vertex computation with limited data
2. authoring (artists don't have to sculpt every vertex)
    - design base shape and make coarse adjustments
3. storage (don't need to remember every vertex in every pose)
4. re-use (apply deformations to different base shapes)

# Animation by Transformation

Translate or rotate...

1. Change each vertex
    - compute N vertices
    - transmit N vertices between CPU and GPU
2. Change a transformation
    - change 1 number (maybe 12 for a matrix)
    - send 1 matrix to GPU

Downside: limited things we can do (with simple transformations)

# Deformation-based shape control

1. Shape Interpolation (Morphing)
2. Non-Linear (complex) deformations
3. Grid Deformers (Free-Form Deformations)
4. Cages

5. Skeletons