CS451Real-time Rendering Pipeline

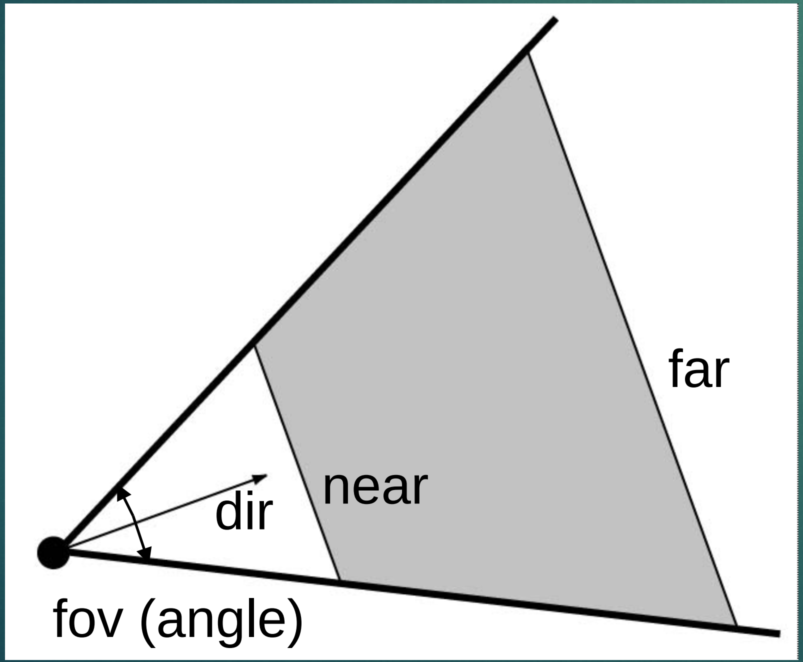
JYH-MING LIEN
DEPARTMENT OF COMPUTER SCIENCE
GEORGE MASON UNIVERSITY

Based on Tomas Akenine-Möller's lecture note

You say that you render a *”3D scene”*, but what does it mean?

- First of all, to take a picture, it takes a camera
  - Decides what should end up in the final image

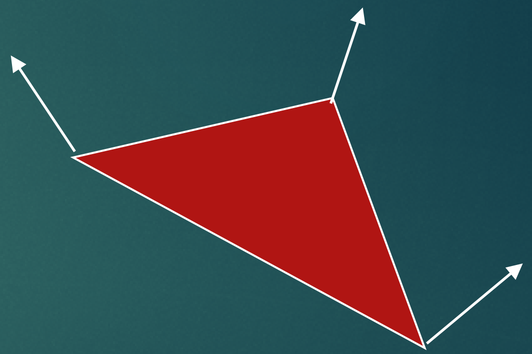
Point camera



- Create image of geometry inside gray region
- Used by OpenGL, DirectX, ray tracing, etc.

You say that you render a *”3D scene”*, but what does it mean?

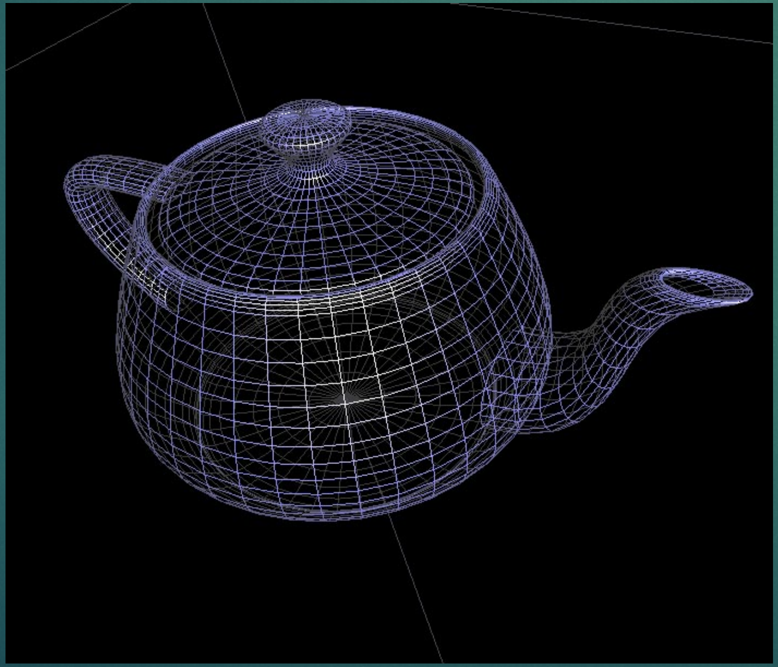
- A 3D scene includes:
  - Geometry (triangles, lines, points, and more)
    - A triangle consists of 3 vertices
      - A vertex is 3D position, and may
  - Material properties of geometry
  - Light sources
  - Textures (images to glue onto the geometry)
- Let us take a look at OBJ format

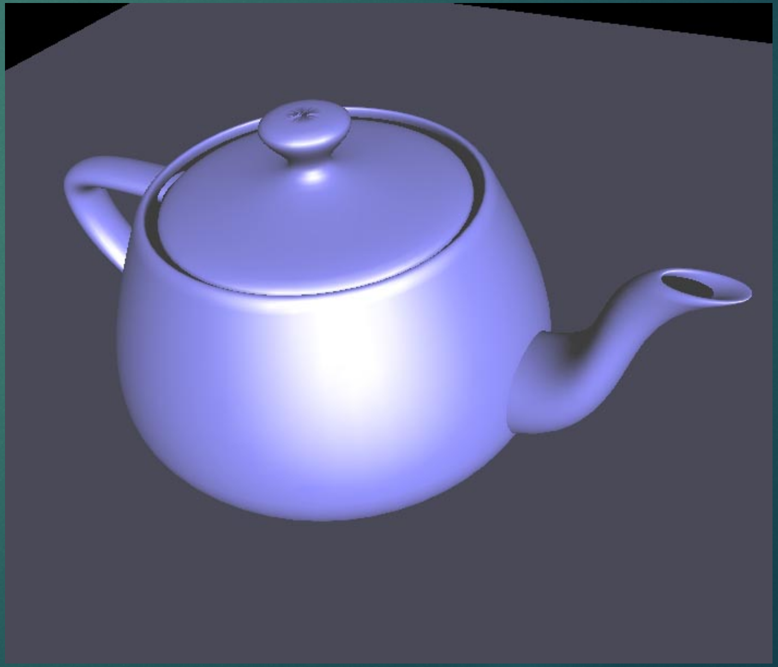


include normals, texture coordinates and more

Rendering Primitives

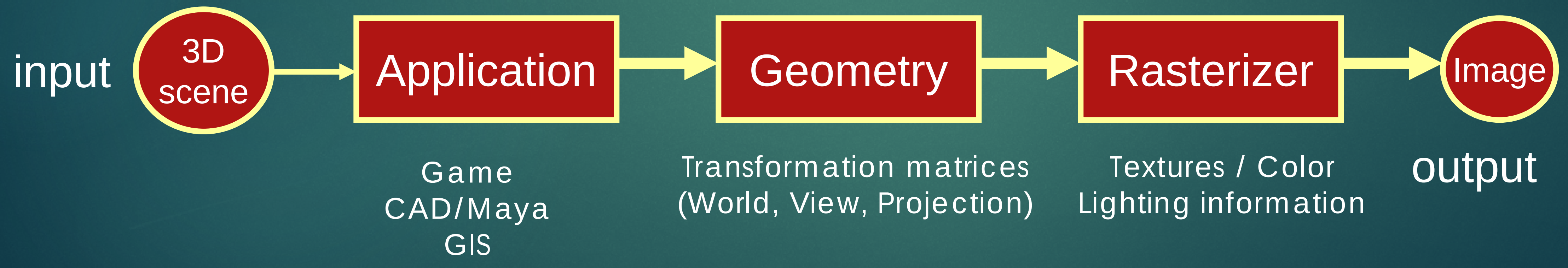
- Use graphics hardware (GPU) for real time computation...
- These GPUs can render points, lines, triangles very efficiently
- A surface is thus an approximation by a number of such primitives

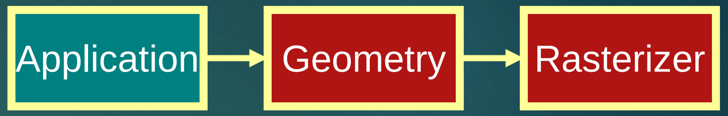




Fixed-Function Pipeline

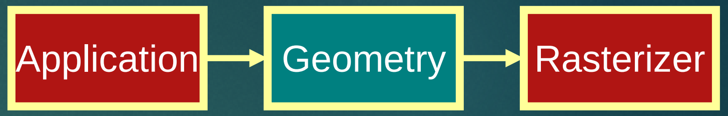
- From chapter 2 in the RTR book
- The pipeline is the "engine" that creates images from 3D scenes
- Three conceptual stages of the pipeline:
  - Application (executed on the CPU)
  - Geometry
  - Rasterizer





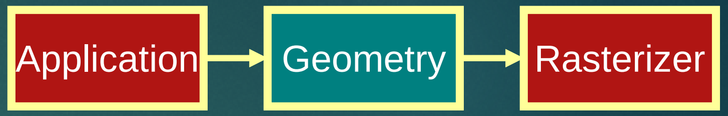
Back to the pipeline:
The APPLICATION stage

- Executed on the CPU
  - Means that the programmer decides what happens here
- Examples:
  - Collision detection
  - Speed-up techniques
  - Animation
- Most important task: send rendering primitives (e.g. triangles) to the graphics hardware



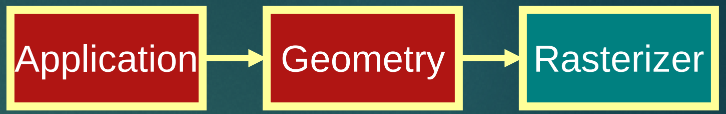
The GEOMETRY stage

- Task: ”geometrical” operations on the input data (e.g. triangles)
- Allows:
  - Move objects (matrix multiplication)
  - Move the camera (matrix multiplication)
  - Compute lighting at vertices of triangle
  - Project onto screen (3D to 2D matrix multiplication)
  - Clipping (remove triangles outside the screen)
  - Map to window



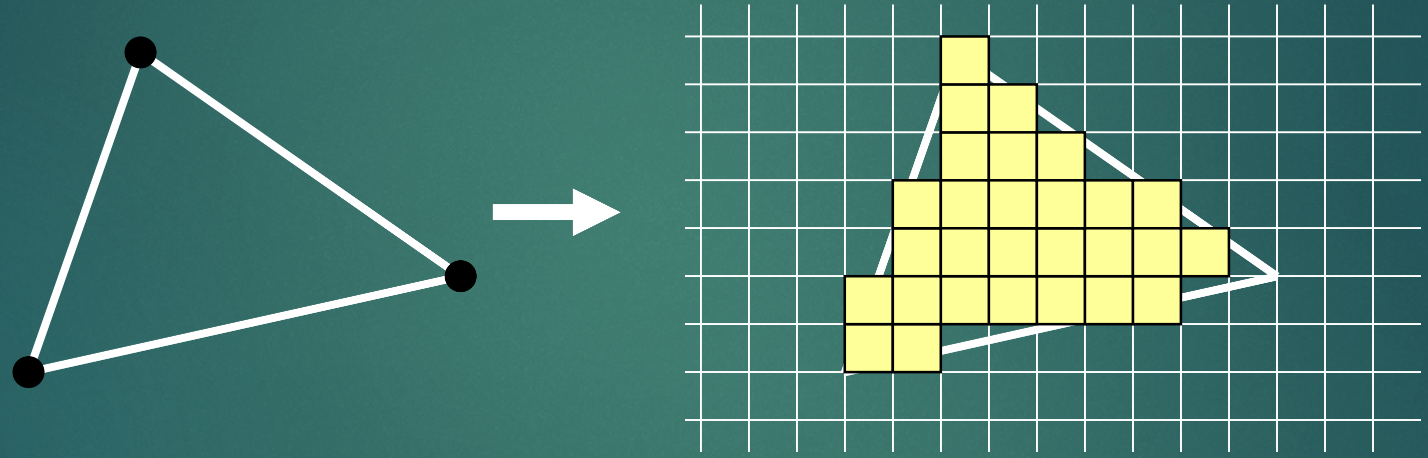
Animate objects and camera

- Can animate in many different ways with 4x4 matrices
- Example:
  - Before displaying a torus on screen, a matrix that represents a rotation can be applied. The result is that the torus is rotated.
- Same thing with camera (this is possible since motion is relative)



The RASTERIZER stage

- Main task: take output from GEOMETRY and turn into visible pixels on screen



- add textures and various other per-pixel operations
- And visibility is resolved here: sorts the primitives in the z-direction

Rewind! Let’s take a closer look

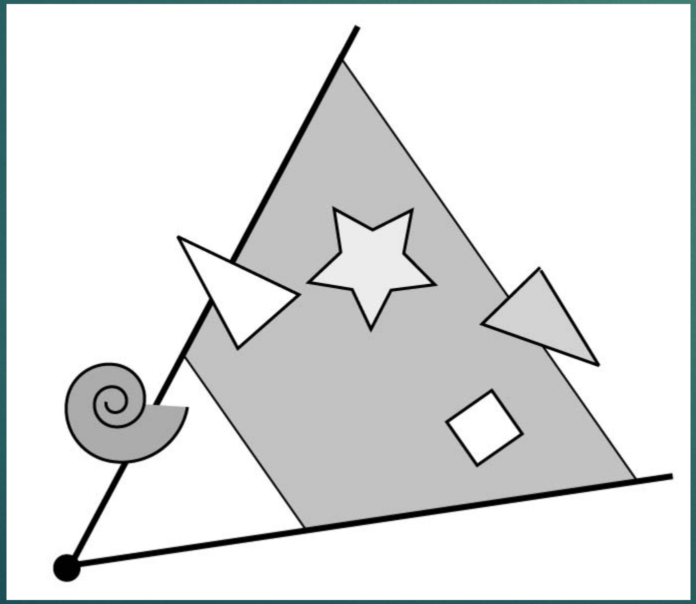
- The programmer ”sends” down primtives to be rendered through the pipeline (using API calls)
- The geometry stage does per-vertex operations
- The rasterizer stage does per-pixel operations
- Next, scrutinize geometry and rasterizer

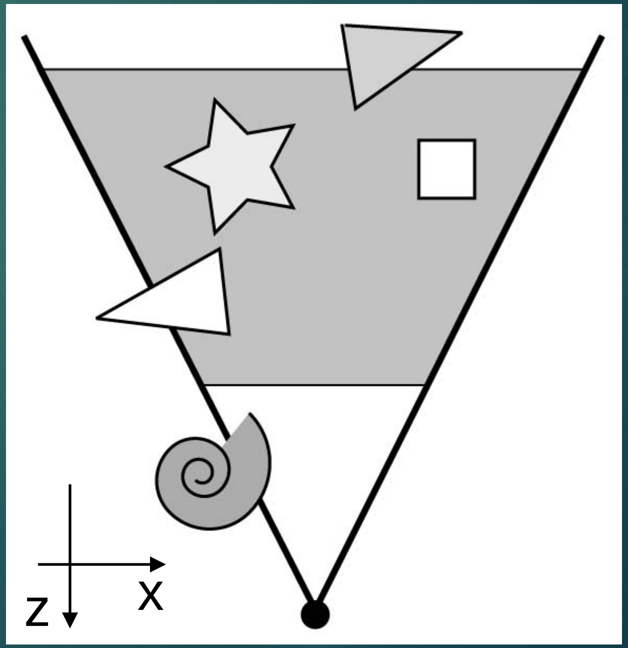
GEOMETRY stage in more detail

- **The model transform**
- Originally, an object is in model space
- Move, orient, and transform geometrical objects into world space
  - Ex: a sphere is defined with origin at (0,0,0) with radius 1
  - Translate, rotate, scale to make it appear elsewhere
- Done per vertex with a 4x4 matrix multiplication
  - How does the matrix look like? Can it be any 4x4 matrix?

The view transform

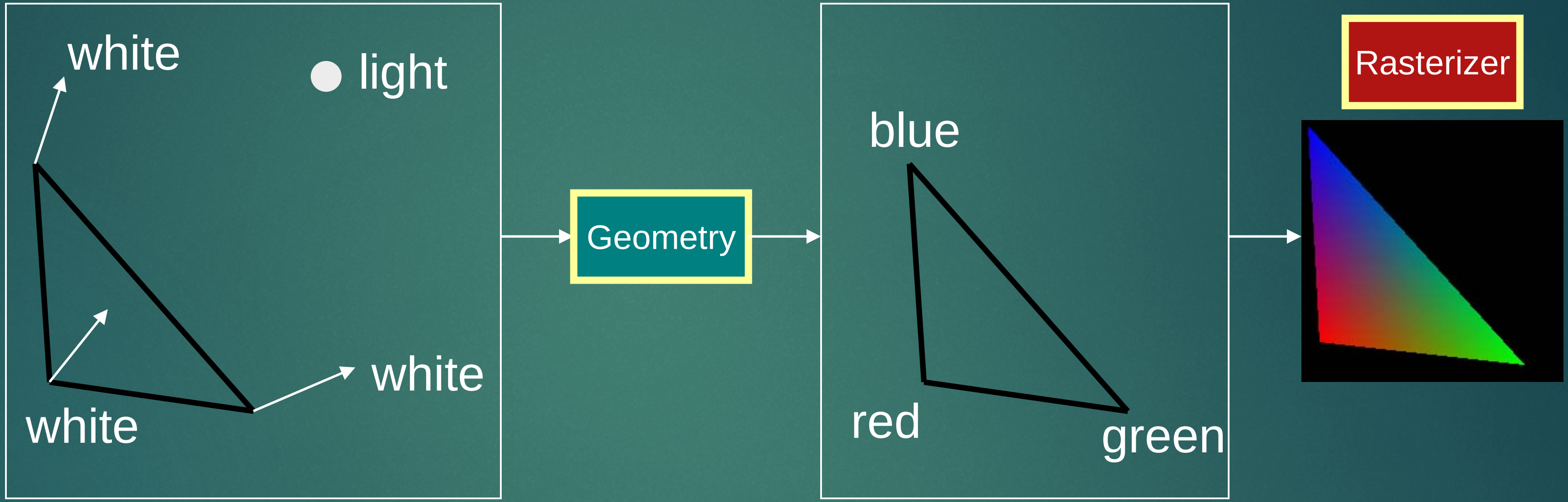
- You can move the camera in the same manner
- But apply inverse transform to objects, so that camera looks down negative z-axis (as in OpenGL)





Lighting

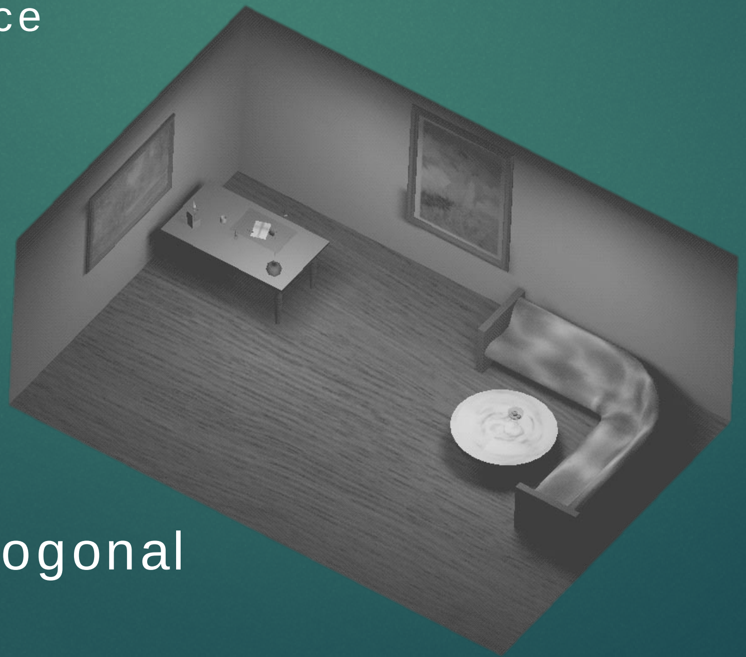
- Compute lighting at vertices



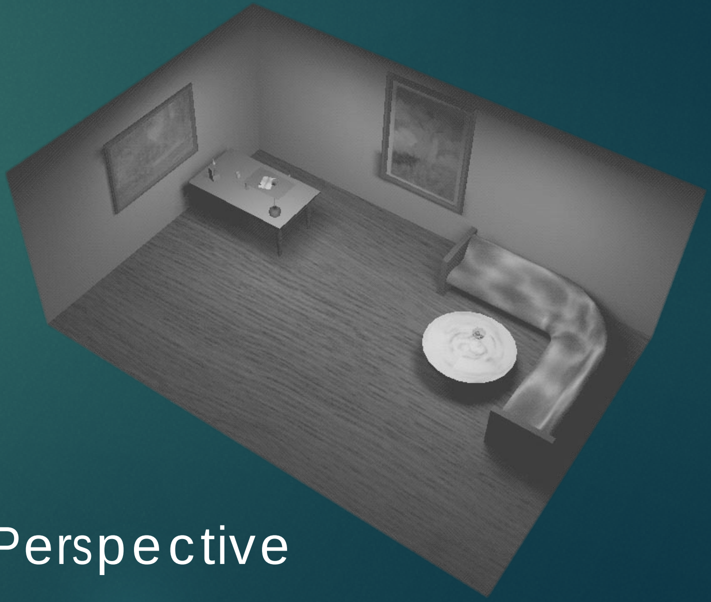
- mimics how light in nature behaves
  - uses empirical models, hacks, and some real theory
- Much more about this in later lectures

Projection

- Two major ways to do it
  - Orthogonal (useful in fewer applications)
  - Perspective (most often used)
    - Mimics how humans perceive the world, i.e., objects' apparent size decreases with distance



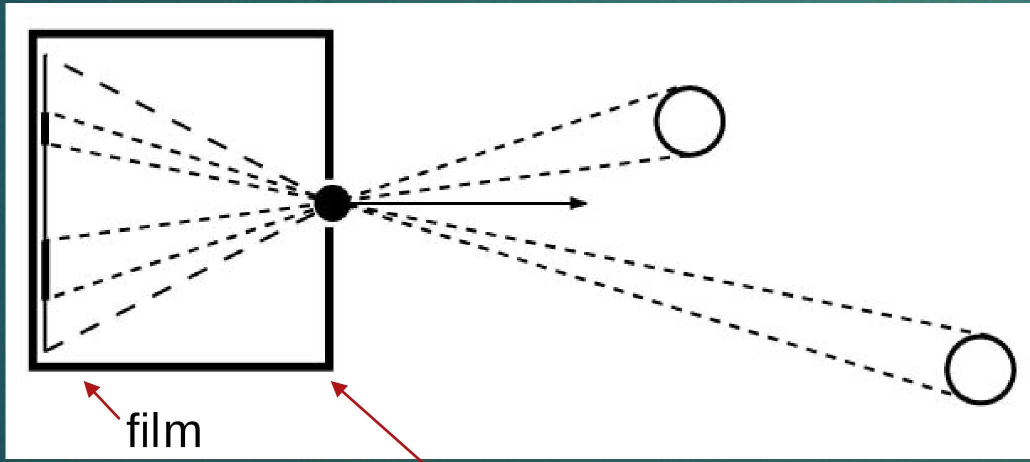
Orthogonal

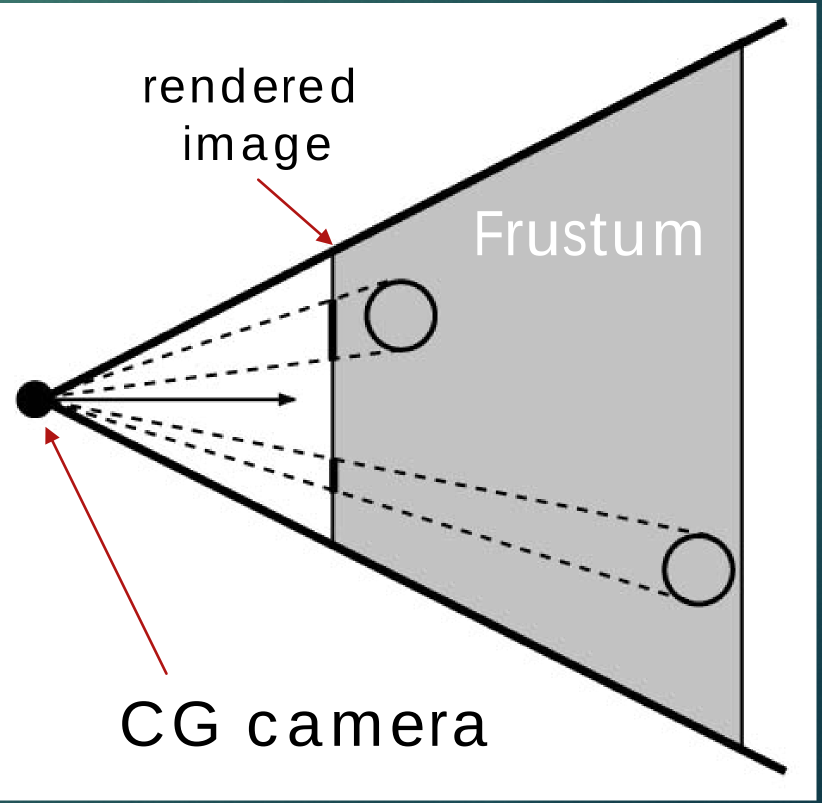


Perspective

Projection

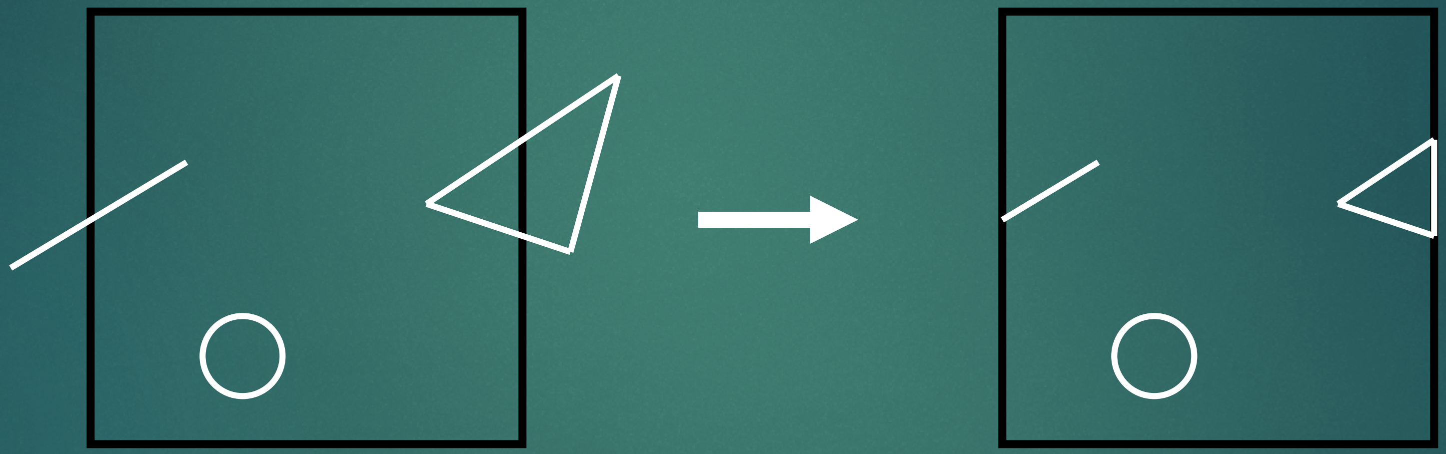
- Also done with a matrix multiplication
- Pinhole camera (left), analog used in CG (right)





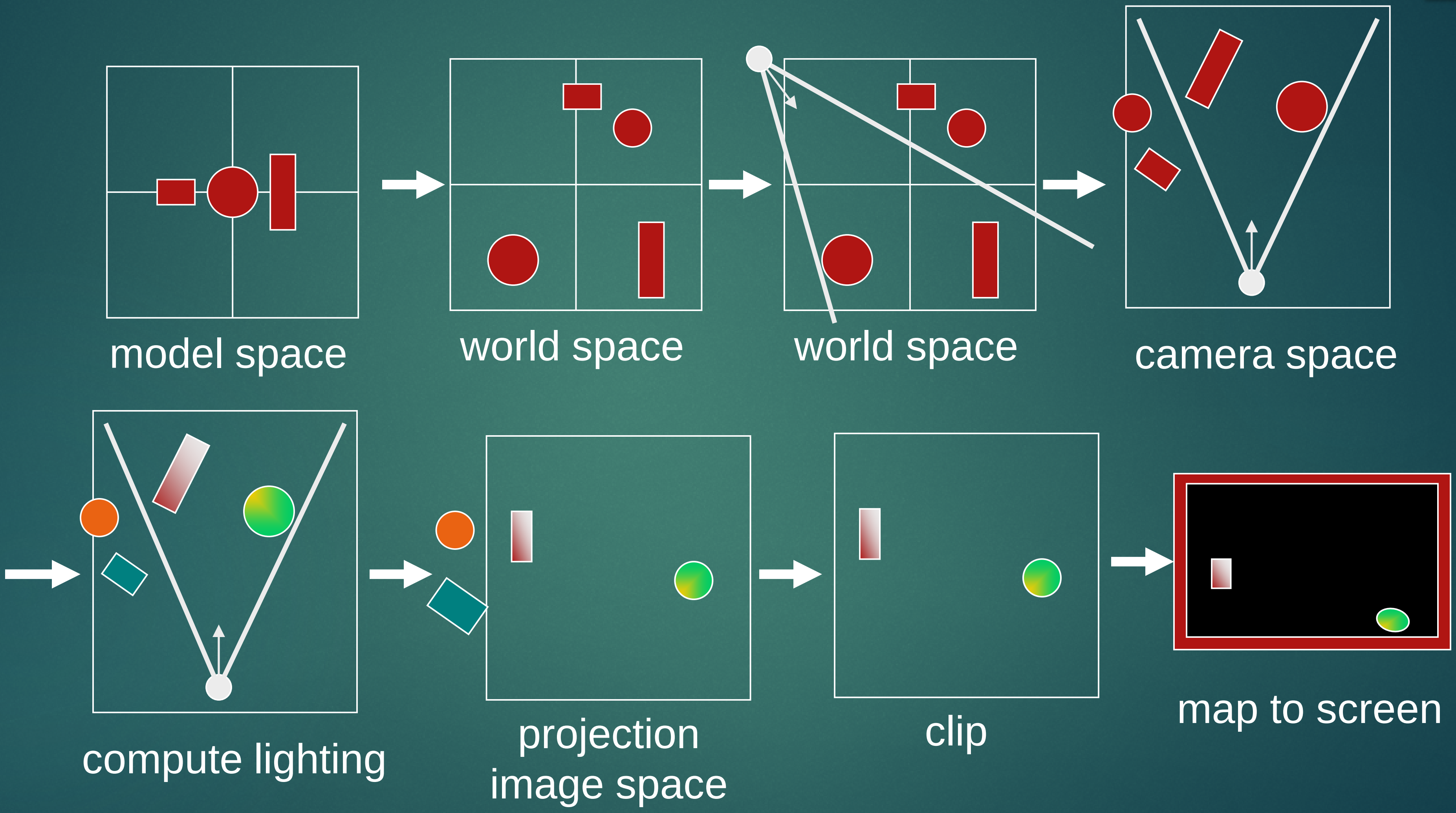
Clipping and Screen Mapping

- Square (cube) after projection
- Clip primitives to square



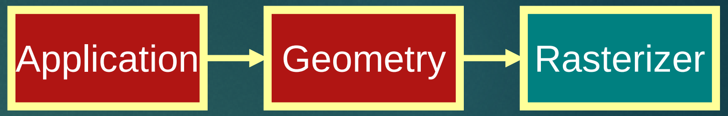
- Screen mapping, scales and translates square so that it ends up in a rendering window
- These screen space coordinates together with Z (depth) are sent to the rasterizer stage

Summary


model space
world space
world space
camera space
compute lighting
projection
image space
clip
map to screen

The RASTERIZER in more detail

- Scan-conversion
  - Find out which pixels are inside the primitive
- Texturing
  - Put images on triangles
- Interpolation over triangle
- Z-buffering
  - Make sure that what is visible from the camera really is displayed
- Double buffering
- And more…

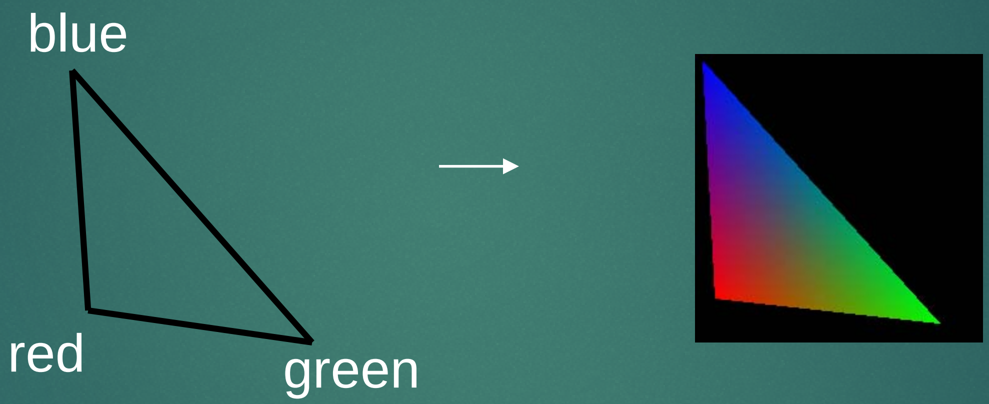


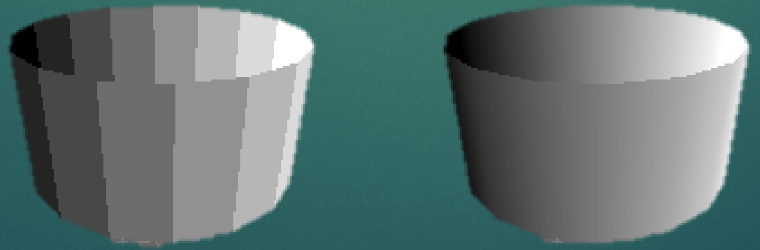
Scan conversion

- Triangle vertices from GEOMETRY is input
- Find pixels inside the triangle
  - Or on a line, or on a point
- Do per-pixel operations on these pixels:
  - Interpolation
  - Texturing
  - Z-buffering
  - And more…

Interpolation

- Interpolate colors over the triangle
  - Called Gouraud interpolation

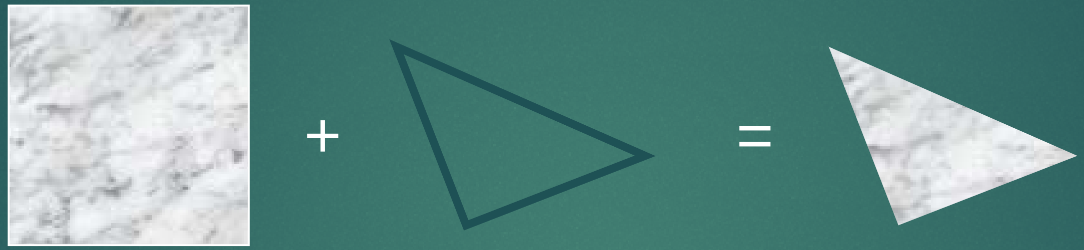




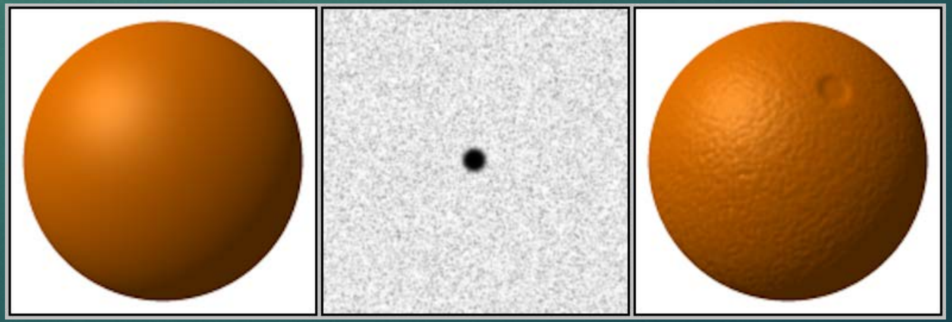
flat Gouraud

Texturing

- texturing is like gluing images onto geometrical object



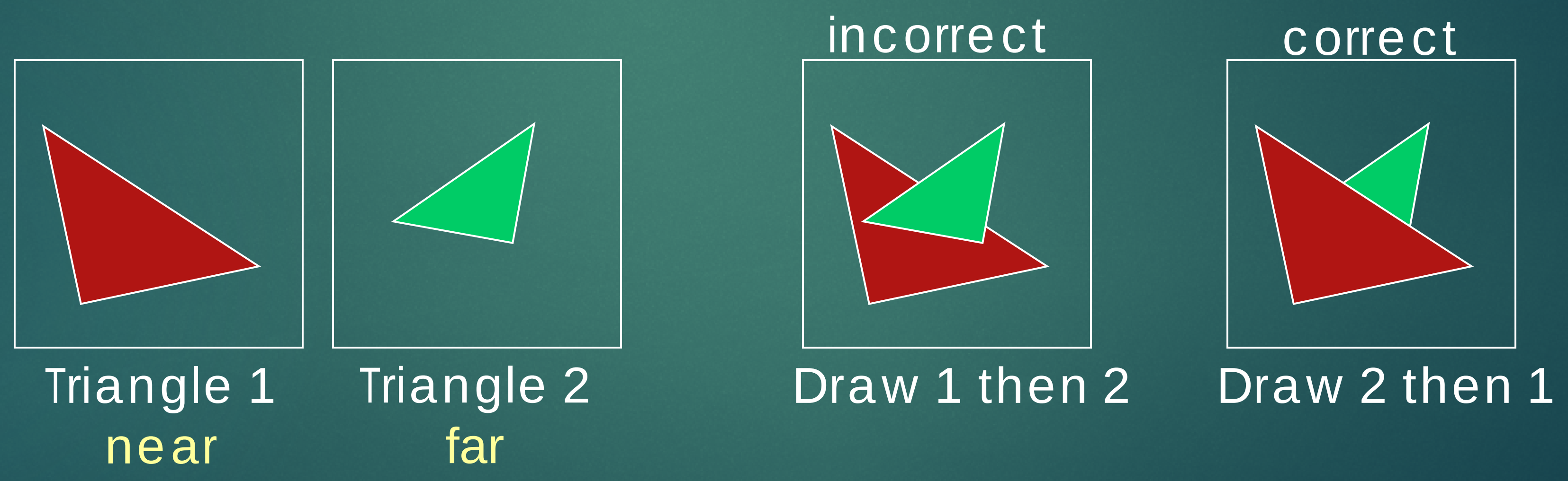
- Uses and other applications
  - More realism
  - Bump mapping
  - Pseudo reflections
  - Light mapping
  - ... many others

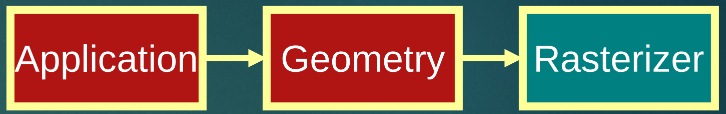


Bump mapping

Z-buffering

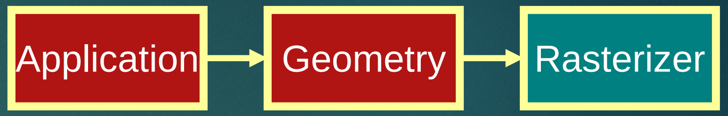
- The graphics hardware is pretty stupid
  - It ”just” draws triangles
- However, a triangle that is covered by a more closely located triangle should not be visible
- Assume two equally large tris at different depths





Z-buffering

- Would be nice to avoid sorting…
- The Z-buffer (aka depth buffer) solves this
- Idea:
  - Store z value (depth) at each pixel
  - When scan-converting a triangle, compute z at each pixel on triangle
  - Compare triangle's z to Z-buffer z-value
  - If triangle's z is smaller, then replace Z-buffer and color buffer
  - Else do nothing
- Can render in any order (if no blending is involved)



Double buffering

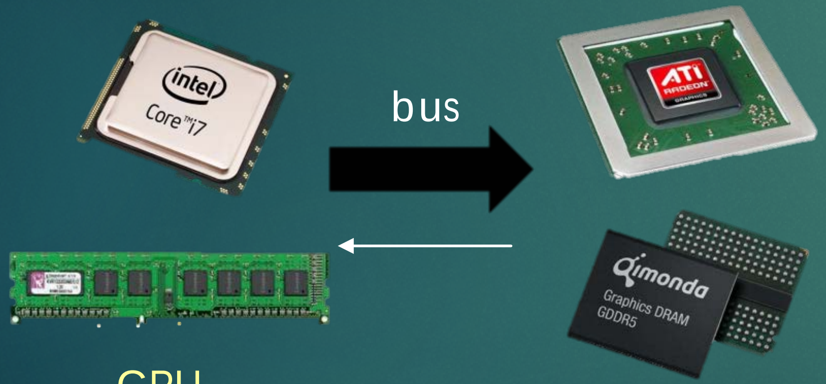
- The monitor displays one image at a time
- So if we render the next image to screen, then rendered primitives pop up
- And even worse, we often clear the screen before generating a new image
- A better solution is ”double buffering”

Double buffering

- Use two buffers: one front and one back
- The front buffer is displayed
- The back buffer is rendered to
- When new image has been created in back buffer, swap front and back

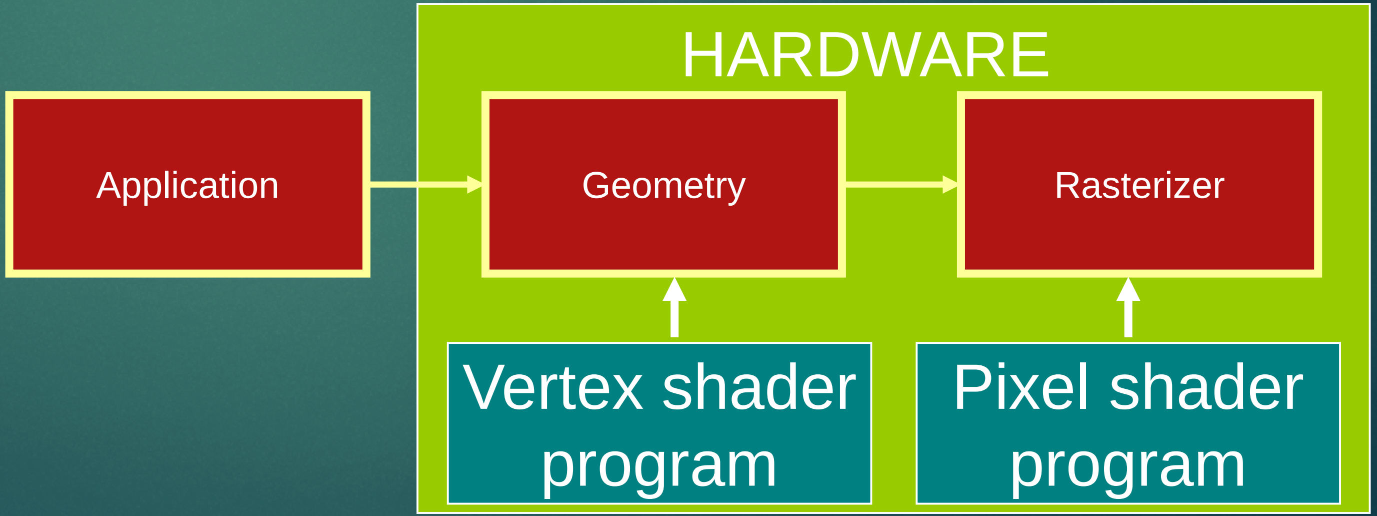
Programmable pipeline

- Programmable shading has become a hot topic
  - Vertex shaders
  - Pixel shaders
  - Adds more control and much more possibilities for the programmer



CPU
Very general
Command GPU

GPU
Highly specialized
e.g, vector calculations
Highly parallelized



More to come when we talk about shaders!!