

ME 290-R:
General Purpose Computation
(CAD/CAM/CAE) on the GPU
(a.k.a. Topics in Manufacturing)

Sara McMains
Spring 2009

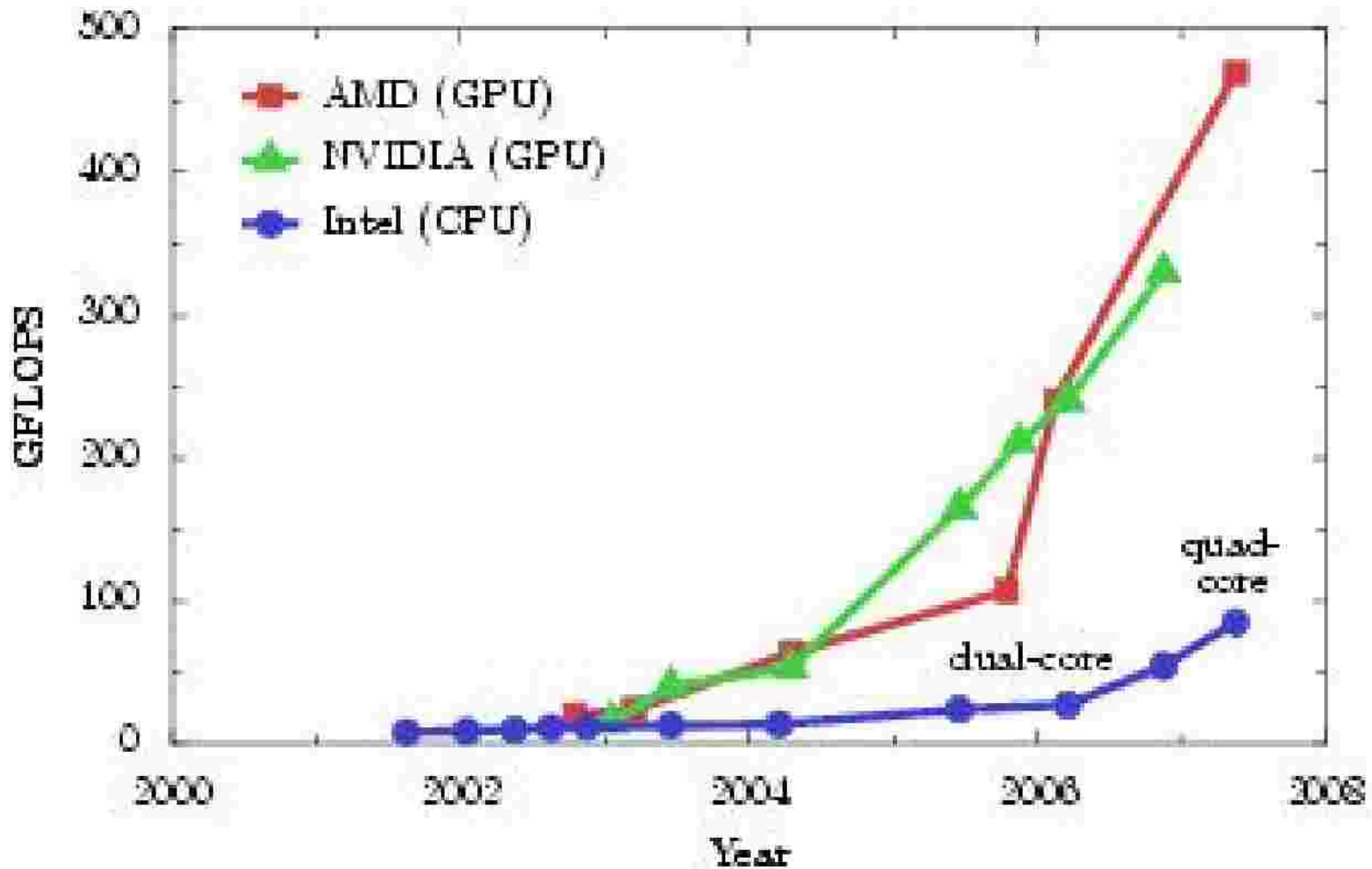
Course Introduction

- The GPU on commodity graphics cards has evolved into an extremely flexible and powerful processor
 - Programmability
 - Precision
- This course will address how to harness that power for general-purpose computation
 - “GPGPU”

Computational Power

- GPUs are fast
- GPUs are getting faster, faster
 - CPUs: 1.4 × annual growth
 - GPUs: 1.7 ×(pixels) to 2.3 × (vertices) annual growth

GPU/CPU Performance



GPU/CPU Performance

- CPU
 - calculating dot product of two vectors
 - taken from Intel's published specs
- GPU
 - using shader (program running on GPU) to execute multiply and add (MAD) instructions
 - MAD4 on 4-component vectors

Growth Rates

- Why are GPUs getting faster so fast?
 - Economics
 - multi-billion dollar video game market
 - Arithmetic intensity
 - specialized nature of GPUs
 - CPUs optimized for sequential code
 - requires cache, branch prediction
 - GPUs optimized for specialized parallel code
 - more transistors used for computation

Flexible and Precise

- Modern GPUs are deeply programmable
 - Programmable pixel, geometry, vertex processors
 - high-level language support
- Modern GPUs support high precision
 - 32 bit floating point throughout the pipeline
 - High enough for many (not all) applications
 - Double precision now available
 - Only 10% of speed
 - On few new cards in hardware
 - Others just emulate

The Potential of GPGPU

- In short:
 - The power and flexibility of GPUs makes them an attractive platform for general-purpose computation
 - Goal: make the inexpensive power of the GPU available to researchers as a computational coprocessor

The Problem

- Can't simply "port" CPU code!
- GPUs designed for & driven by video games
 - Programming model unusual
 - Programming idioms tied to computer graphics
 - Programming environment tightly constrained
- Underlying architectures are:
 - Inherently parallel
 - Rapidly evolving (even in basic feature set!)
 - Largely secret

Course Overview

- Background to understand graphics pipeline
 - graphics operations are performed in a particular order described as the graphics processing pipeline
- A detailed introduction to general-purpose computing on graphics hardware
- Research papers on latest GPGPU applications
 - emphasis on those relevant to CAD/CAM/CAE

Course Prerequisites

- I assume
 - Some experience programming
- I don't assume
 - Prior computer graphics programming experience
- Target audience
 - Engineering grad students whose research involves computation
 - Interested in making that computation faster on the GPU
 - Computer graphics grad students interested in GPU applications beyond graphics

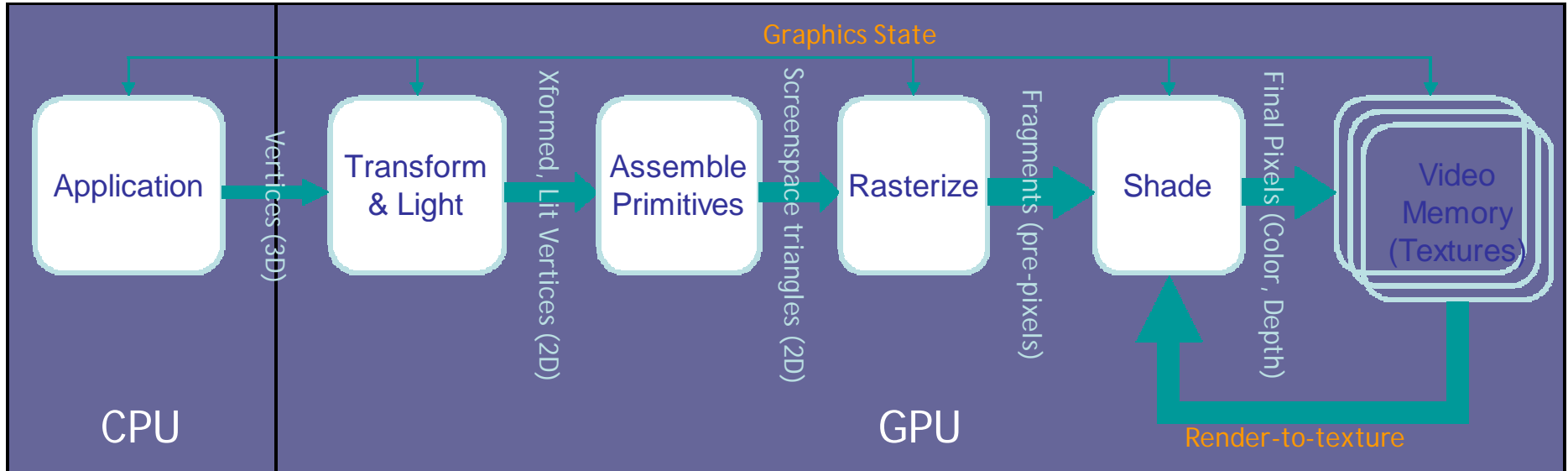
Course Topics

- Introduction to graphics hardware: capabilities, trends, strengths and weaknesses; graphics pipeline introduction.
- Projective geometry: 2D scales, rotations, and translations using homogeneous coordinates; 3D scales, rotations, and translations.
- Projection: orthographic projection, perspective projection, vanishing points.
- Primitive assembly and rasterization, clipping, antialiasing; visibility, interpolation, shading, and texture mapping.
- Raster operations: scissor, alpha, stencil, and depth testing; blending and logic operations.
- The streaming model; high level programming for GPUs
- Bottleneck identification, debuggers
- Depth Peeling and Projective Textures
- Reductions, Sorting, Searching on the GPU
- CUDA
- OpenCL

Course Topics: Applications

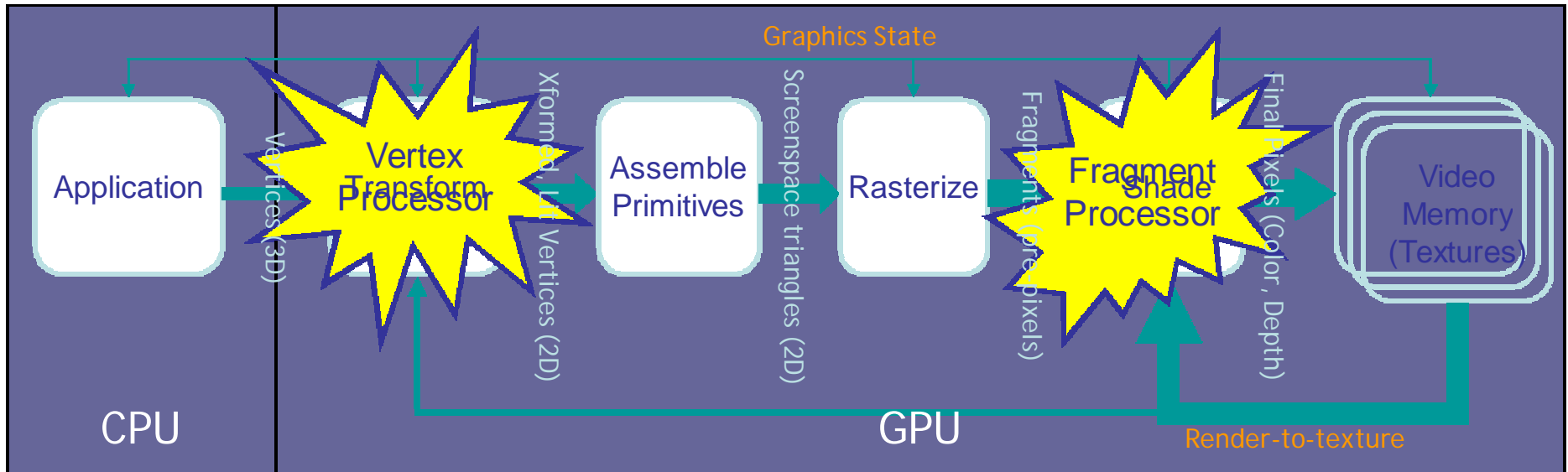
- GPU fluid visualization and simulation
- Sparse matrix solvers, conjugate gradient, and multigrid
- Dense linear solvers
- PDE solving
- Image processing
- Boolean operations, constructive solid geometry, CSG evaluation
- Manufacturability analysis
- Distance fields
- Splines
- Ray tracing

The Traditional Graphics Pipeline



- A simplified graphics pipeline

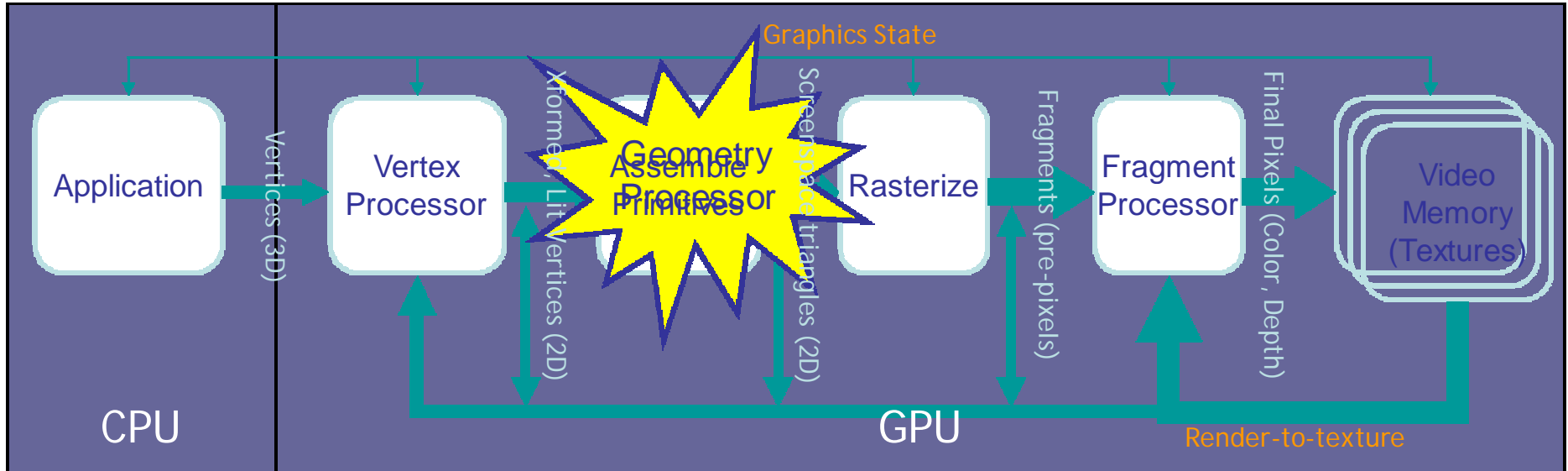
The Semi-Modern Graphics Pipeline



- Programmable vertex processor

- Programmable pixel processor

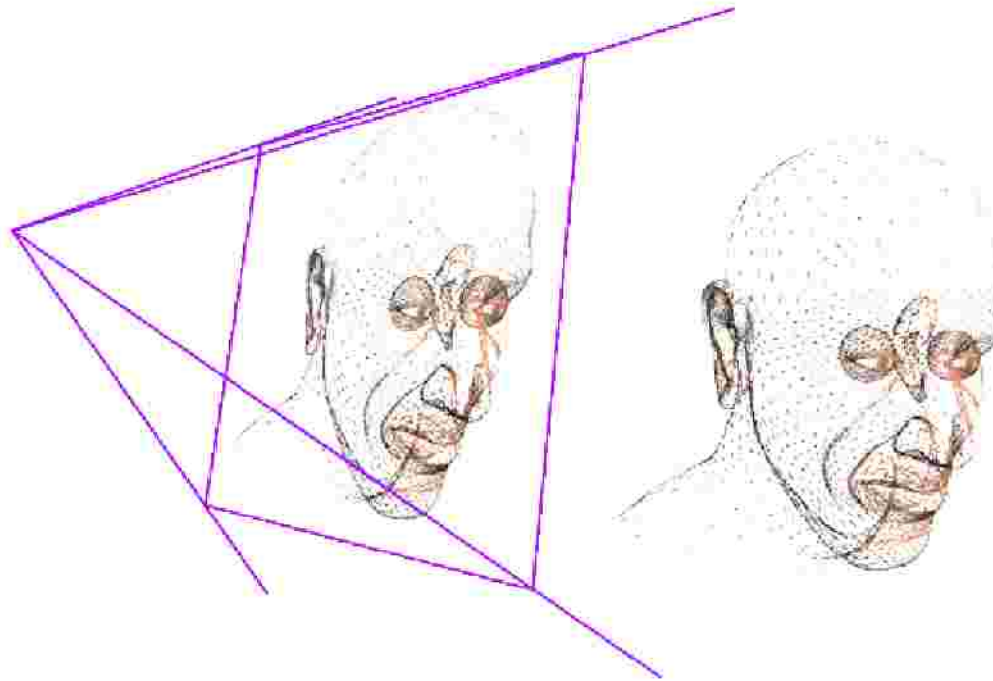
The Modern Graphics Pipeline



- Programmable primitive assembly!
- More flexible memory access!

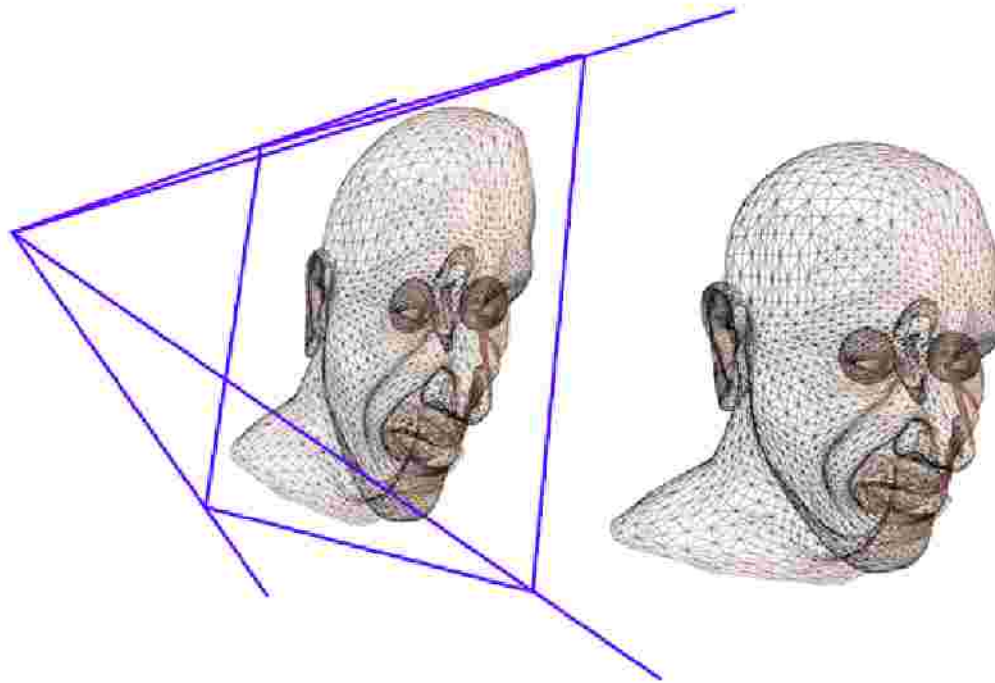
GPU Pipeline: Transform

- Vertex processor (multiple in parallel)
 - Transform vertices from “world space” to “image space”
 - Compute per-vertex lighting



GPU Pipeline: Assemble

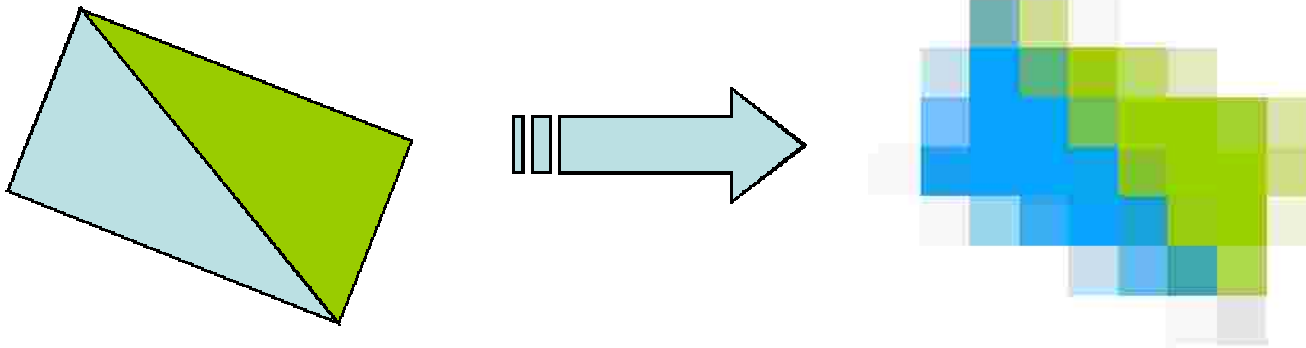
- Assemble vertices into triangles



GPU Pipeline: Rasterize

- Rasterizer

- Determine screen position covered by each triangle
 - potential pixel, an image “fragment”
 - Pixel + associated data: color, depth (distance), etc.
- Interpolate per-vertex quantities across pixels



GPU Pipeline: Shade

- Fragment processors (multiple in parallel)
 - Compute a color for each pixel
 - Optionally read colors from textures (images)



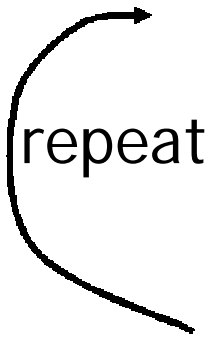
GPGPU

- Typically used fragment processors
 - scenes usually have more fragments than vertices
 - hence more fragment processors than vertex, geometry processors
 - e.g. 6 vertex processors vs. 16 fragment processors on NVIDIA's 6800 Ultra
 - Today GPUs have unified architecture

GPGPU

- Typical GPGPU programming
 - divide application into individual parallel sections
 - implemented in fragment programs
 - input, output are data arrays
 - set up a big quadrilateral that covers as many pixels as output size
 - run fragment program on each pixel in parallel
 - read from arbitrary memory
 - write only to corresponding pixel location
 - read final pixels (slow) or use “occlusion queries” to get number of pixels written (fast)

repeat



Acknowledgements

- Ian Buck
- Trond Hagen
- Naga Govindaraju
- Mark Harris
- Aaron Lefohn
- David Luebke
- John Owens
- Timothy Purcell