EDA075

Mobile Computer Graphics

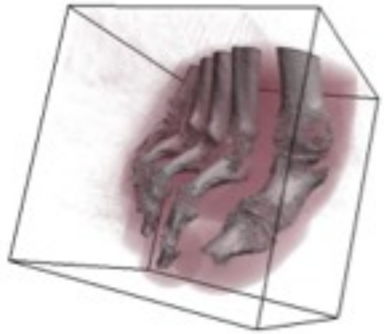
Michael Doggett
Department of Computer Science
Lund University

© 2009 Tomas Akenine-Möller and Michael Doggett

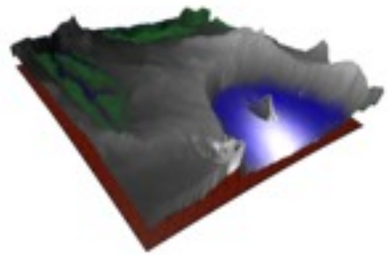
My Background

My Background

- Hardware design
  - Volume Rendering Hardware



  - Displacement Mapping

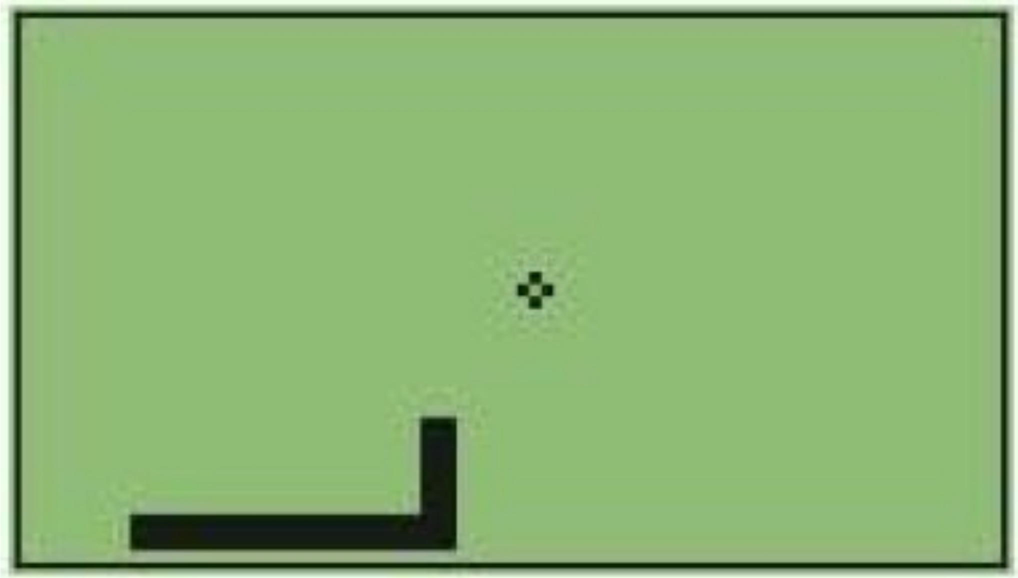


  - GPUs - ATI
    - Xbox360
    - Radeon 2xxx-5xxx

© 2009 Tomas Akenine-Möller and Michael Doggett

Quiz: which game is the world's most played electronic game?

- Halo? Mario? Final Fantasy ...? World of Warcraft? Tetris?



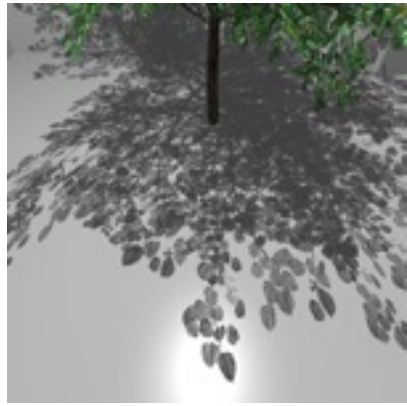
- It might be "Snake"
- According to The Guardian: "it took Nintendo 10 years to sell 100M Game Boys whereas Nokia sold 128M handsets last year alone (2003)"

© 2009 Tomas Akenine-Möller and Michael Doggett

Introduction

- LUGG=Lund University Graphics Group
  - Magnus Andersson
  - Björn Johnsson
  - Jim Rasmusson
  - Lennart Ohlsson
  - Petrik Clarberg
  - Jakob Munkberg
  - Tomas Akenine-Möller
  - Research
    - Mobile computer graphics
    - Shadows and visibility
    - Rasterization algorithms
    - Ray tracing-based algorithms
    - Graphics hardware

Intel Lund
- Larrabee



© 2009 Tomas Akenine-Möller and Michael Doggett

Why mobile graphics?

- Phone is not just a phone!
  - Calendar, camera, messages, images, animations, games, surfing, email, sounds+music, radio, tv, addresses, notes etc.
- BIG market: ~1 billion mobile phones/year (2007/09)
  - Phones 9% decrease, smartphones 13% increase -Q109
  - Only games on mobiles: $300Mn in 2006 (est.)
  - Mobile games: $7.2Bn market in 2010 (Informal)
- The ***visual*** is a strong differentiating factor

© 2009 Tomas Akenine-Möller and Michael Doggett

Do the visual well, and your device will sell

© 2009 Tomas Akenine-Möller and Michael Doggett

Displays

- Used to be one-bit graphics @ ~50x80 pixels
- Today 16-18 bits is common
  - Satio has 24 bits
- Resolution:
  - Today ~208x176 – 800x480...
  - QVGA (320x240) is the norm...
  - Nokia
    - series 90 is 640x320, N95 is 320x240
    - N900 800x480
  - Sony Ericsson
    - P990, M600i, K800i: 320x240
    - Satio 640x360
  - Apple Iphone is 480x320
- We'll get 1024x768 in the future...
- Makes graphics possible!

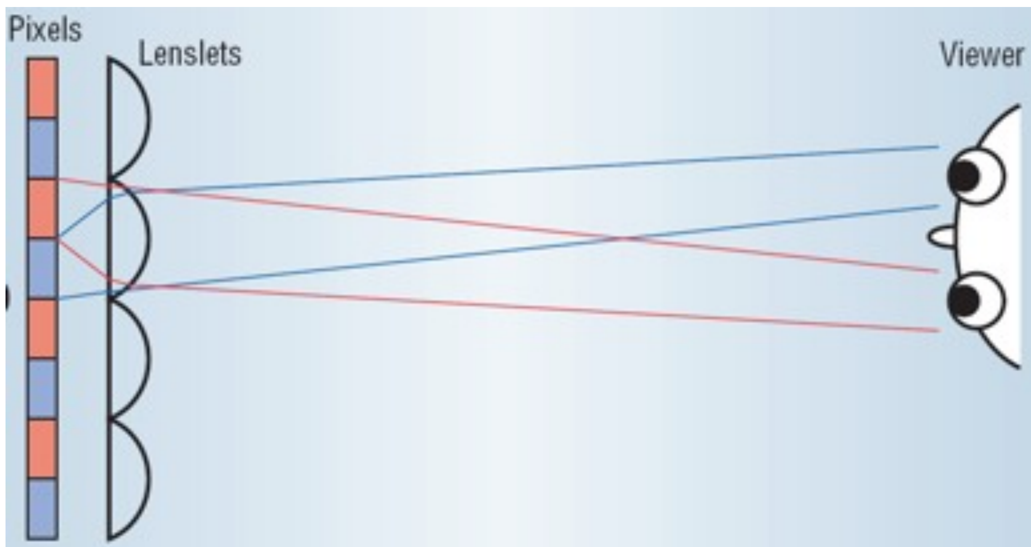


© 2009 Tomas Akenine-Möller and Michael Doggett

Displays in the near future?

- Real 3D displays are around the corner
- Large increase in 3D movies, Ice Age, Up, ...
- Big breakthrough might be mobiles
- Simple principle:

Stereo principle



© 2009 Tomas Akenine-Möller and Michael Doggett

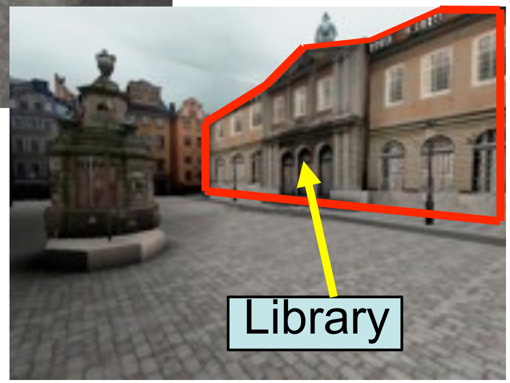
Costly with 3D graphics on 3D displays

- There are displays with, e.g., 9 views

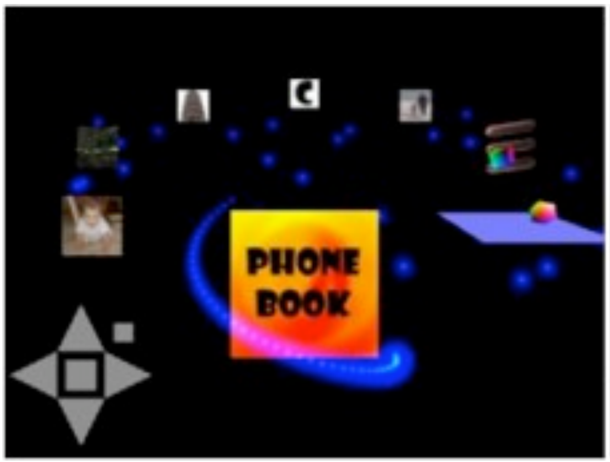
© 2009 Tomas Akenine-Möller and Michael Doggett

Some examples where 3D graphics key technology

Maps:

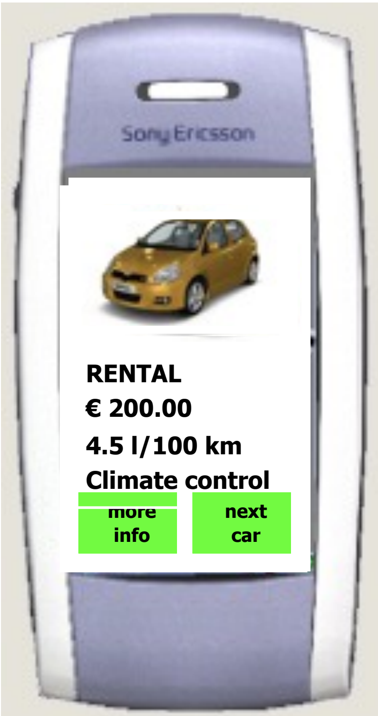


User interfaces (simpler, smoother, more intuitive)



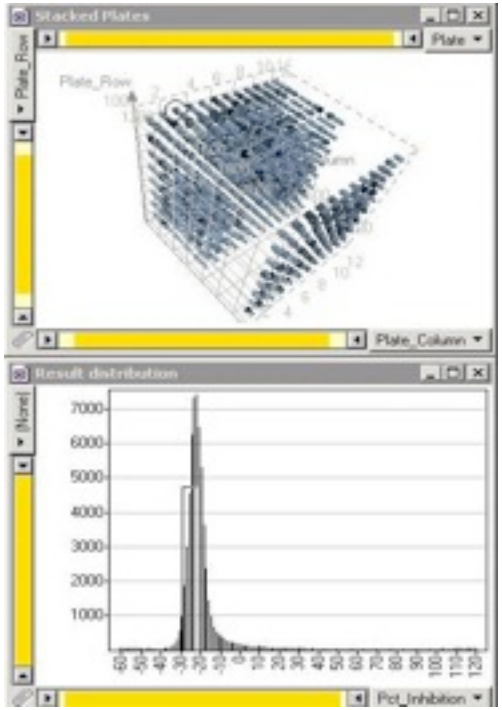
Simple stuff: screen savers

More applications...



E-commerce

Data mining/visualization?



Gaming, game development

Copyright 2005, by Interactive Brains, Co., Ltd.

More?

You decide!

© 2009 Tomas Akenine-Möller and Michae

Slide copyright by Kari Pulli, Nokia

Mobile graphics applications

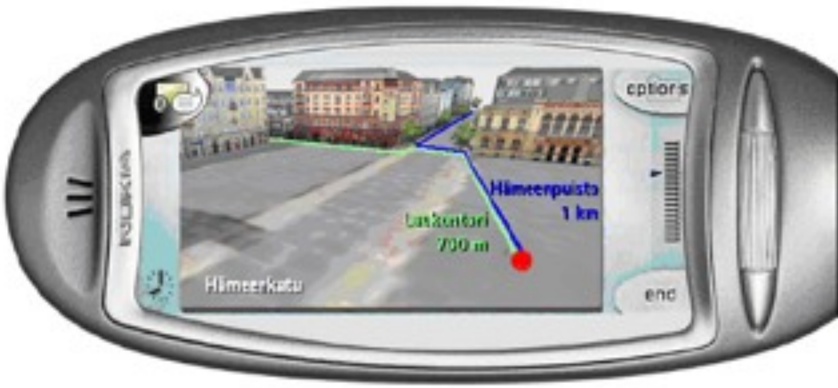
SIGGRAPH2005

3D Menu

3D Games

3D Animation

3D Messaging



Location services

Advertising

iPhone/iPod Touch Apps

- 1.5 billion downloads
- In 1 year, 65,000 apps
- 100,000 memberships of dev program

Rolando 2 by ngmoco

Flight Control and Real Racing by Firemint

Source: Edge magazine October 2009

© 2009 Tomas Akenine-Möller and Michael Doggett

Mobile 3D Graphics Hardware

- OpenGL ES 1.0
  - 100s millions of phones
- OpenGL ES 2.0
  - Imagination Technologies PowerVR SGX
    - iPhone 3GS/iPod Touch
    - Sony Ericsson Satio
    - Nokia N900
  - Nvidia Tegra
    - ZuneHD

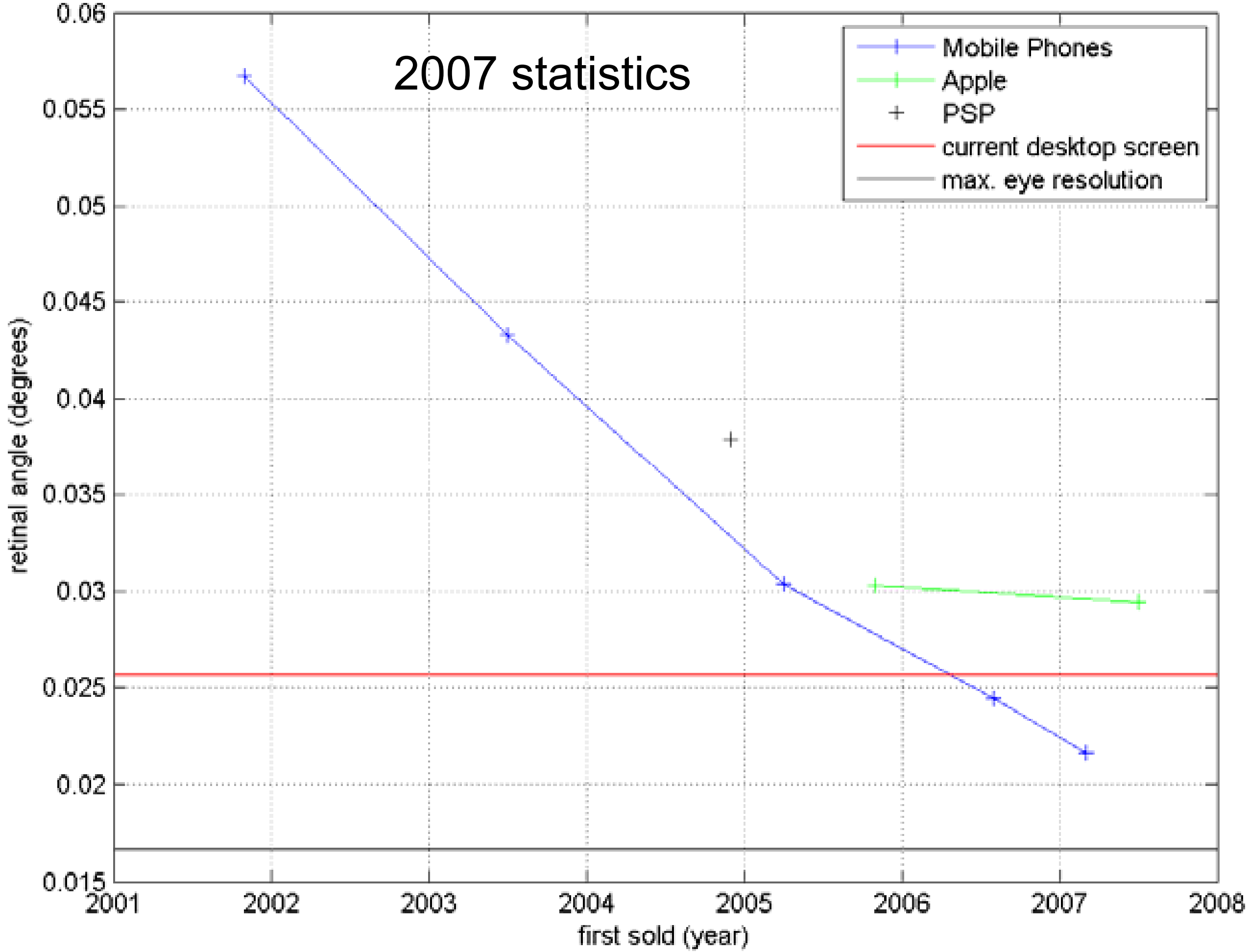
© 2009 Tomas Akenine-Möller and Michael Doggett

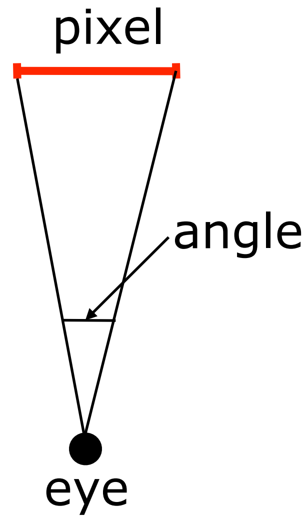
Why is it hard to do 3D graphics on mobile devices?

- Small amount of memory
- Limited instruction set
- Low clock frequency
  - 100-200 MHz ARM9
  - 400-600 MHz ARM11
  - 600 MHz ARM CortexA8
- Small area on the chip for CG
- Must be cheap and physically small
- Powered by batteries!
  - A memory access is one of the most expensive operations
  - Battery growth: 9% per year
    - Performance growth: 40% per year

Small display, but very close to eyes

- Our measurements [in 2003]:
  - Average eye-to-pixel angle is 1-4 times larger for mobile than for a laptop/desktop





Still, about the same requirements as for desktop (where resolution has increased as well)

So, we need about the same image quality as for desktop graphics

Our mobile graphics research (1)

- We have an improved variant called ”Ericsson Texture Compression” (used to be called ***i*PACKMAN**)
  - Is an ”optional extension” in OpenGL ES. Supported by PowerVR SGX in iPhone 3GS

- Jacob Ström from Ericsson Research will be here for one lecture to talk about (all) texture compression schemes

© 2009 Tomas Akenine-Möller and Michael Doggett

Our mobile graphics research (2)

- We are also doing:
  - Buffer compression (color/depth)
  - HDR texture compression
  - Culling mechanisms
  - Stereo rendering
  - and more...

© 2009 Tomas Akenine-Möller and Michael Doggett

Info about the course

- EDA075 Mobile Computer Graphics
  - It is really quite a bit about graphics hardware too
- 7.5 points *(in my mind, the project is 3p, the rest 4.5p)*
- How to fulfill the course requirements:
  - Two programming assignments (C++)
    - 2 persons per group
  - A small project (C++) - 2 options
    - 2 persons per group
    - Write a 2-4 page report on what you did
  - Pass the written exam
- There will be guest lecturers in this course
  - Jacob Ström, Ericsson Research
  - Carl Loodberg, Illusion Labs
  - Jim Rasmusson, Ericsson Mobile Platforms
  - Erik Månsson, TAT
- Literature: no book – instead research papers + some new material.
  - Could be of interest for people that have not worked much with graphics: ”Real-Time Rendering”, 2008, by Akenine-Möller, Haines, and Hoffman

© 2009 Tomas Akenine-Möller and Michael Doggett

More info about the course

- Two parts:
  1. APIs and how to write applications today on a mobile phone
  2. Graphics hardware for mobiles
     - We will learn about the underlying algorithms used in hardware, and not so much about the hardware itself!
       - Algorithms are interesting! The rest is implementation!
     - Also performance analysis

© 2009 Tomas Akenine-Möller and Michael Doggett

Two programming assignments

- A1: SceneGraph
  - Hi-level C++ API for 3D graphics
  - We will use new iPod Touch 32GB
- A2:
  - Implement parts of a software rasterizer
  - Measure memory bandwidth
    - Improve
  - Same framework could be used in the project

© 2009 Tomas Akenine-Möller and Michael Doggett

About the project

- Two persons per project
- Two different paths (do one of them):
  - P1: iPod Touch/iPhone app
    - Make a cool game, application, or demo
    - Use Assignment 1 framework
  - P2: SW Rasterization
    - Minimize memory bandwidth given a certain amount of onchip memory
    - More challenging! Possibility to create new algorithms. Invent!

- Competition! (more info later)
  - P1: a jury will decide in December who wins
  - P2: the group that uses least memory bandwidth to render a given scene wins!
- You must write a short (2-4 page) report
- Time to start thinking about nice projects now...
  - You need to clear projects of type P1 with me. Write ½ page and send as an email.

© 2009 Tomas Akenine-Möller and Michael Doggett

M3G projects from 2005

- Winner: ***Low rider***
  - by Magnus Borg and Erik Zivkovic
- Look at Stanford iPhone course for ambitious ideas

Course schedule

- W1
  - **Intro** (today)
  - **Mobile API Overview** **[Release of A1]**
- W2
  - **A1 seminar/lecture: SceneGraph Framework (Magnus Andersson)**
  - **How to rasterize a triangle and interpolate** **[Release of A2]**
- W3
  - **A2 seminar/lecture, fixed math, texturing+mipmap+tcache+framework** **[Show solution A1]-Pluto Lab**
  - **Texture Compression (Jacob Ström, Ericsson Research)** **[Project start]**
- W4
  - **Performance analysis + Buffer compression + Zmin+Zmax-culling** **[Show solution A2]-Pluto Lab**
  - **Real-time buffer compression**
- W5
  - **(Carl Loodberg, Illusion Labs), Mobile phones (Jim Rasmusson, Ericsson Mobile Platforms)**
  - **Existing graphics architectures**
- W6
  - **Antialiasing + 3D User interfaces (Erik Månsson, TAT)**
- W7
  - **Competition + project finished + Summary** **[Projects finished]**

© 2009 Tomas Akenine-Möller and Michael Doggett

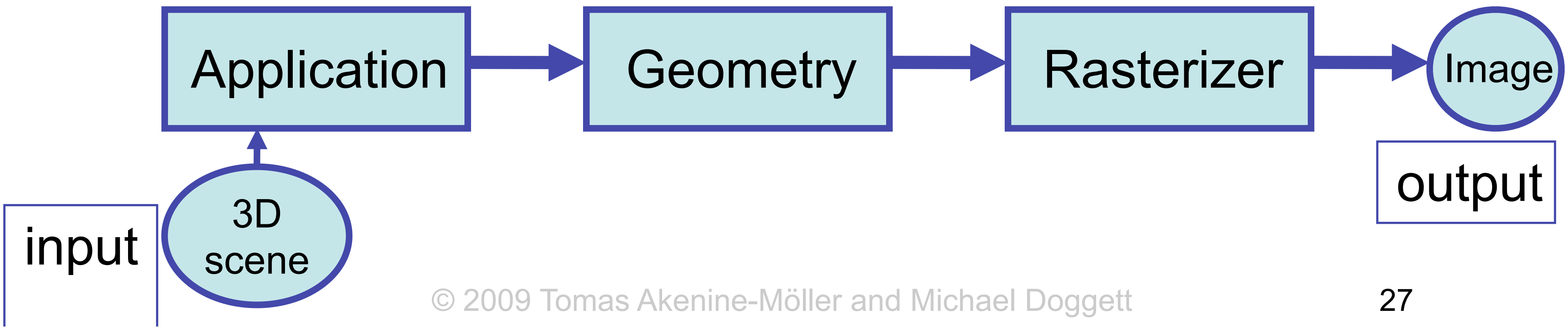
2nd hour of intro lecture

- Quick overview of real-time graphics
- Sign up on thursday
- Competition is optional
- SceneGraph framework ‘should’ work on linux, mac

© 2009 Tomas Akenine-Möller and Michael Doggett

The Real-Time Rendering Pipeline

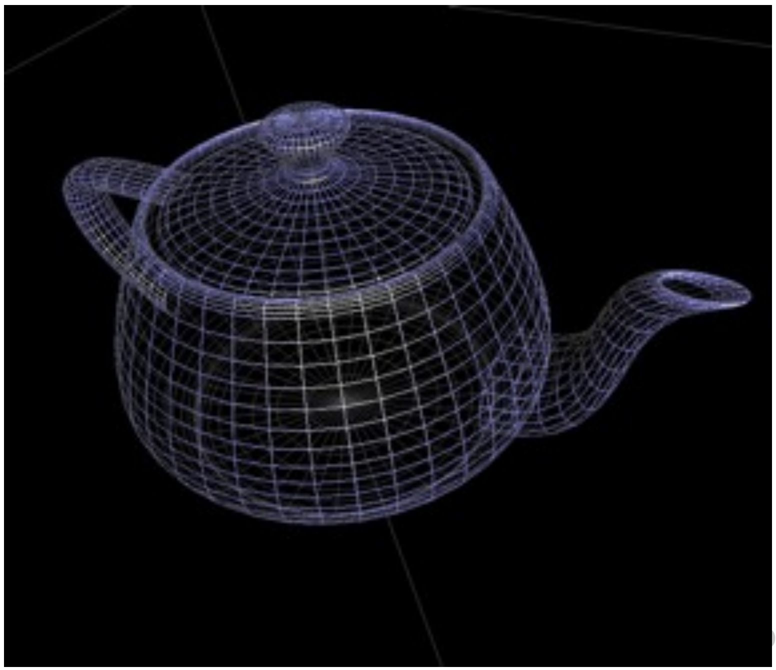
- [Chapter 2 in the the Real-Time Rendering book, which is not required]
- The pipeline is the "engine" that creates images from 3D scenes
- Three conceptual stages of the pipeline:
  - Application (executed on the CPU)
  - Geometry
  - Rasterizer



© 2009 Tomas Akenine-Möller and Michael Doggett

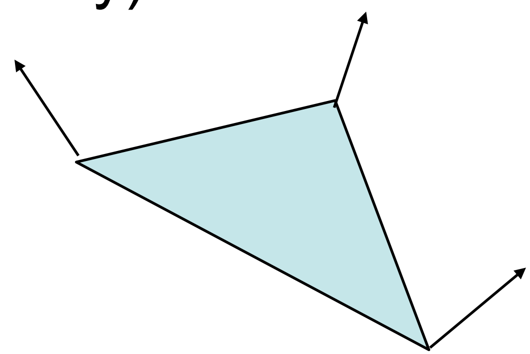
Rendering Primitives

- Use graphics hardware for real time…
  - Though, mobile phones have either software rendering, or dedicated hardware, or a mix
- The available APIs can render points, lines, triangles.
  - For mobiles: OpenGL ES (embedded systems)
- A surface is thus an approximation by a number of such primitives.



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

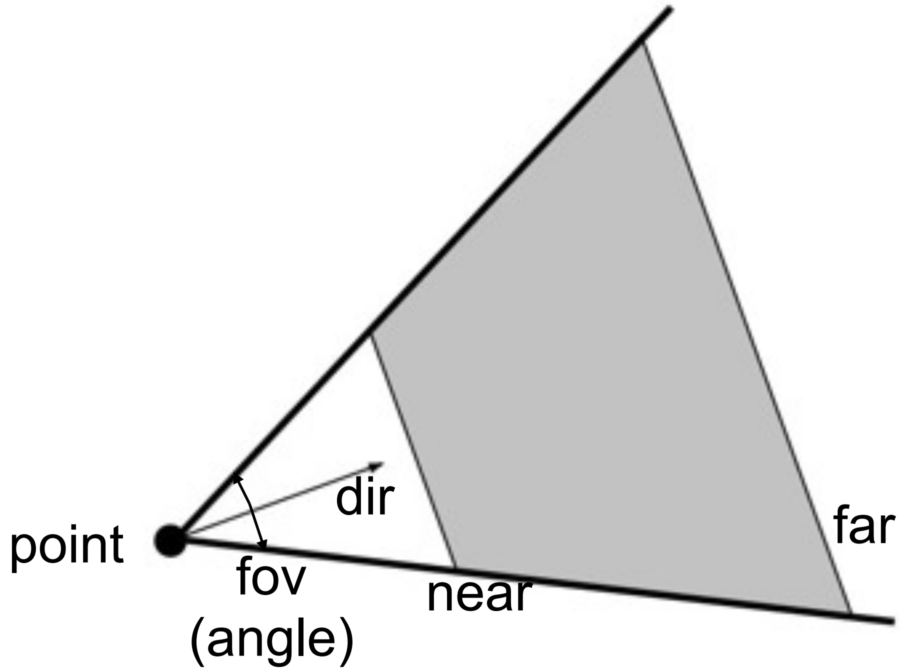
- First, of all to take a picture, it takes a camera – a virtual one.
  - Decides what should end up in the final image
- A 3D scene is:
  - Geometry (triangles, lines, points, and more)
  - Light sources
  - Material properties of geometry
  - Textures (images to glue onto the geometry)
- A triangle consists of 3 vertices
  - A vertex is 3D position, and may include normals and more.



© 2009 Tomas Akenine-Möller and Michael Doggett

Virtual Camera

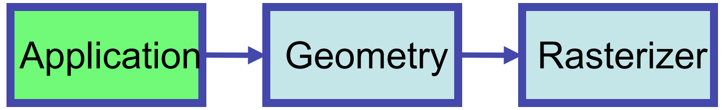
- Defined by position, direction vector, up vector, field of view, near and far plane.



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

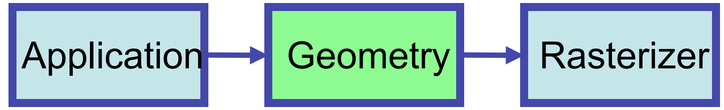
© 2009 Tomas Akenine-Möller and Michael Doggett

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) through the graphics API (which then executes in SW or HW)

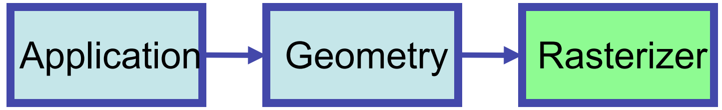
© 2009 Tomas Akenine-Möller and Michael Doggett



The GEOMETRY stage

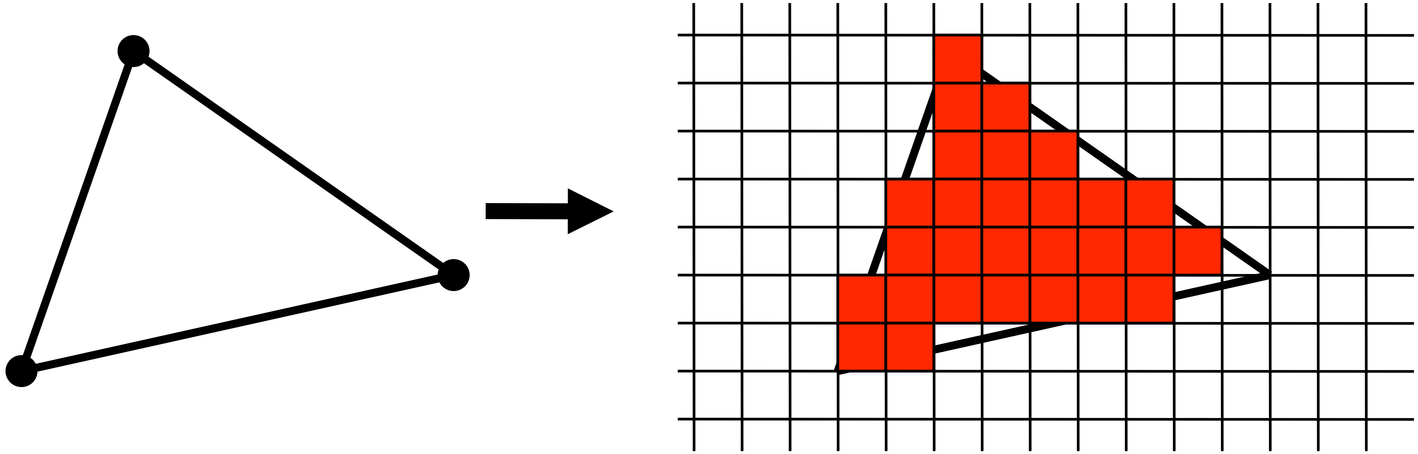
- Task: ”geometrical” operations on the input data (e.g. triangles)
- Allows:
  - Move objects (matrix multiplication)
  - Move the camera (matrix multiplication)
  - Compute lighting at vertices of triangle
  - Project onto screen (3D to 2D)
  - Clipping (avoid triangles outside screen)
  - Map to window
  - Vertex shaders (allows the developer to do arbitrary tasks per vertex)

© 2009 Tomas Akenine-Möller and Michael Doggett



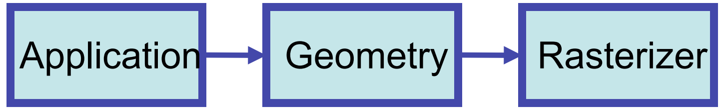
The RASTERIZER stage

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



- Also, add textures and various other per-pixel operations
- And visibility is resolved here: sorts the primitives in the z-direction
- Pixel shaders (also called fragment shaders)

© 2009 Tomas Akenine-Möller and Michael Doggett

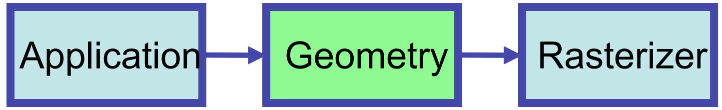


Rewind! Let’s take a closer look

- 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

© 2009 Tomas Akenine-Möller and Michael Doggett

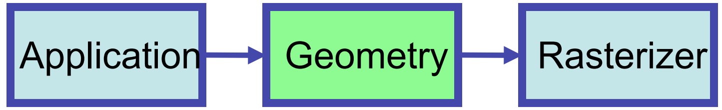
The GEOMETRY stage in more detail



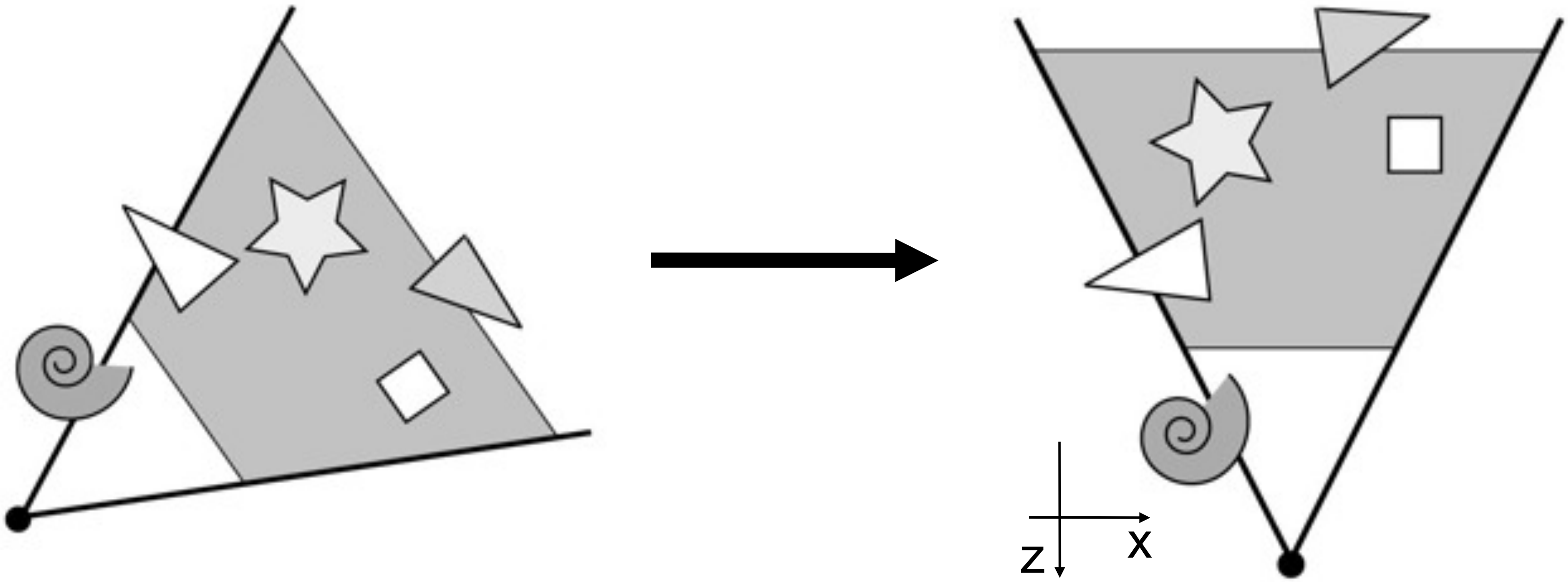
- **The model transform**
- Originally, an object is in ”model space” or ”object space”
- Move, orient, and transform geometrical objects into ”world space”
- Example, a sphere is defined with origin at (0,0,0) with radius 1
  - Translate, rotate, scale to make it appear elsewhere
- Done per vertex with a 4x4 matrix multiplication!
- The user can apply different matrices over time to animate objects

© 2009 Tomas Akenine-Möller and Michael Doggett

GEOMETRY
The view transform

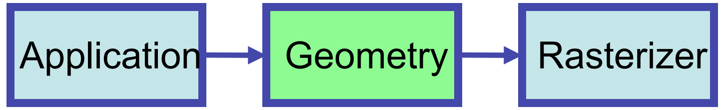


- You can move the camera in the same manner
- But apply inverse transform to objects, so that camera looks down negative z-axis

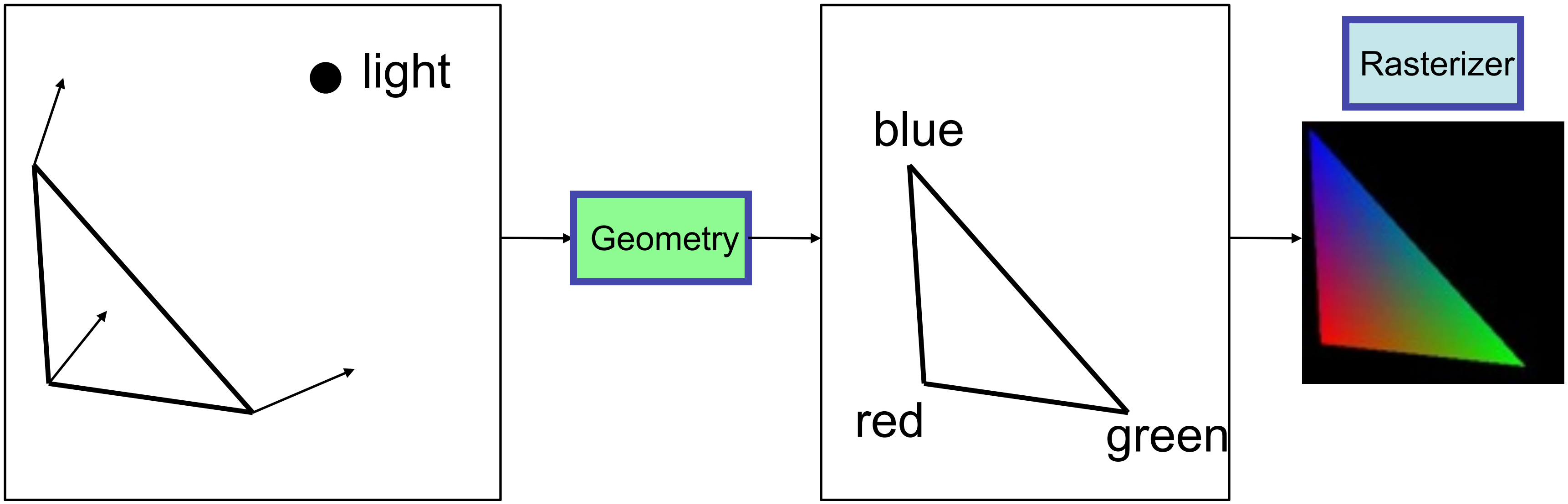


© 2009 Tomas Akenine-Möller and Michael Doggett

GEOMETRY
Lighting



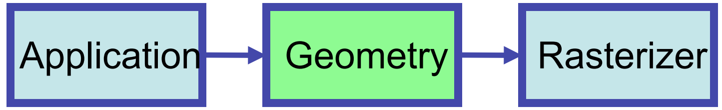
- Compute ”lighting” at vertices



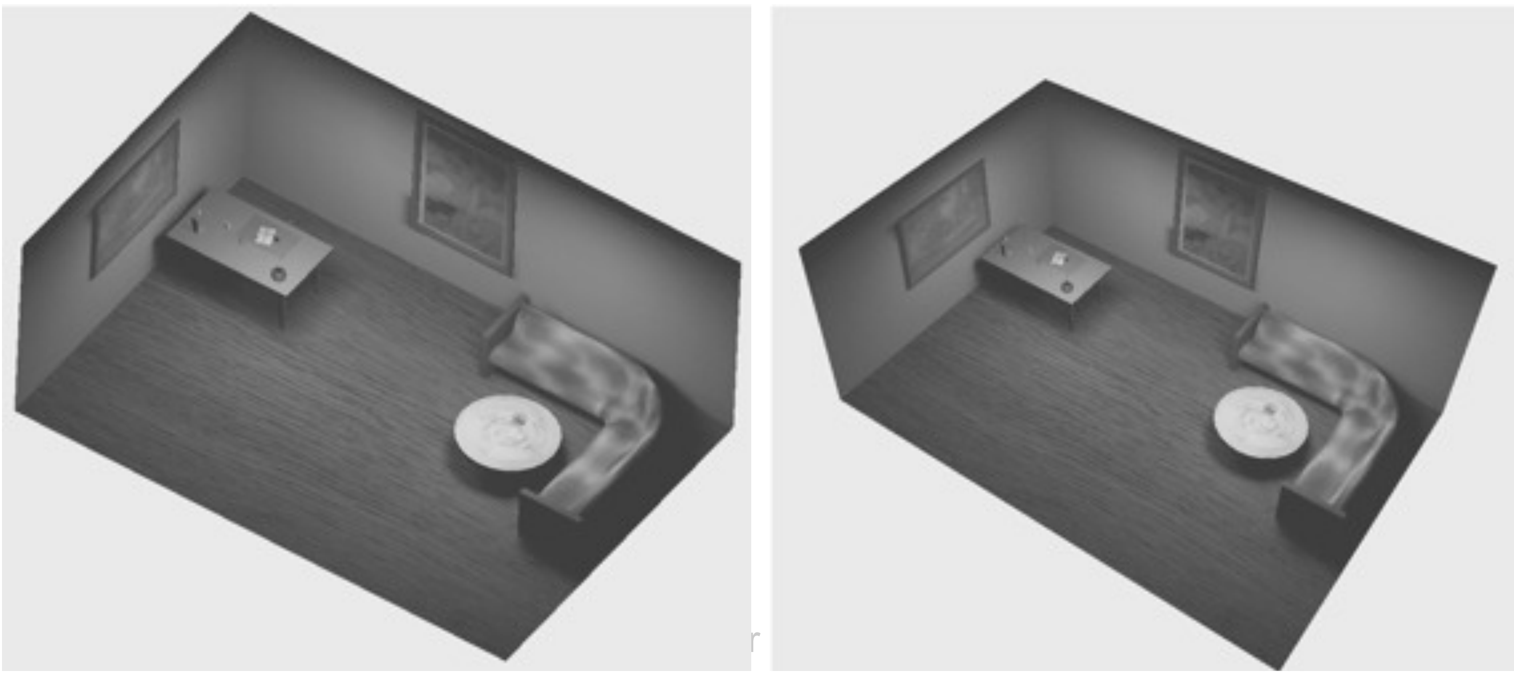
- Try to mimic how light in nature behaves
  - It’s hard to use empirical models, so use hacks, and some real theory

© 2009 Tomas Akenine-Möller and Michael Doggett

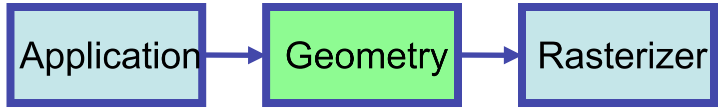
GEOMETRY
Projection



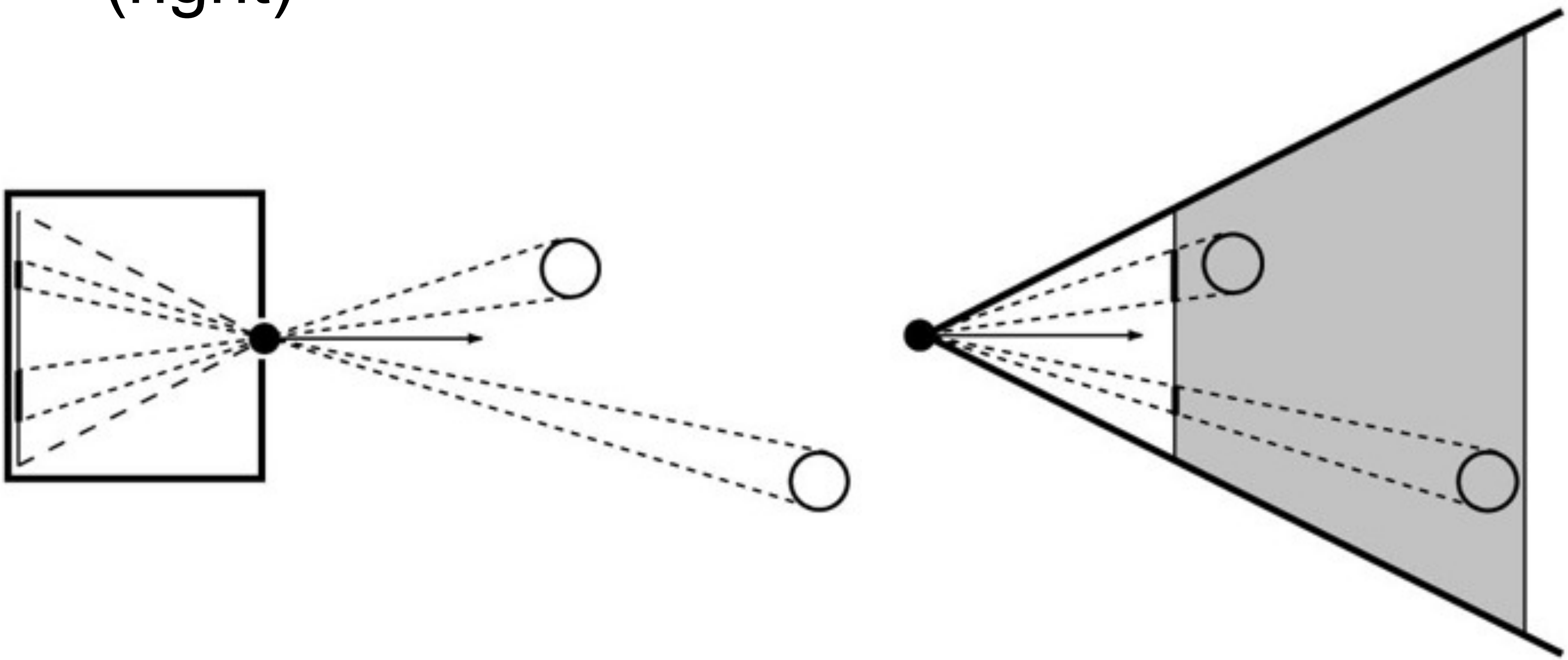
- Two major ways to do it
  - Orthogonal (useful in few applications)
  - Perspective (most often used)
    - Mimics how humans perceive the world, i.e., objects' apparent size decreases with distance



GEOMETRY Projection



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



© 2009 Tomas Akenine-Möller and Michael Doggett

Ultraquick review of homogeneous notation

- Why? $$\begin{pmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{pmatrix} \mathbf{v} = \mathbf{M}\mathbf{v} = \mathbf{v} + \mathbf{t},$$

- Solution: $\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix}$ and in general: $\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \\ v_w \end{pmatrix}$

Using homogenous coordinates, translation becomes:

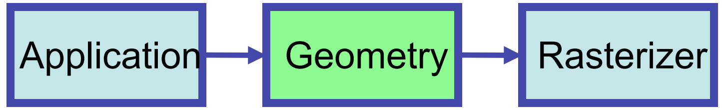
$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{v} = \begin{pmatrix} v_x + t_x \\ v_y + t_y \\ v_z + t_z \\ 1 \end{pmatrix} = \mathbf{v} + \mathbf{t}$$

- Projection:

$$\mathbf{M}\mathbf{v} = \mathbf{h} = \begin{pmatrix} h_x \\ h_y \\ h_z \\ h_w \end{pmatrix} \Longrightarrow \begin{pmatrix} h_x/h_w \\ h_y/h_w \\ h_z/h_w \\ h_w/h_w \end{pmatrix} = \begin{pmatrix} h_x/h_w \\ h_y/h_w \\ h_z/h_w \\ 1 \end{pmatrix} = \mathbf{p}$$

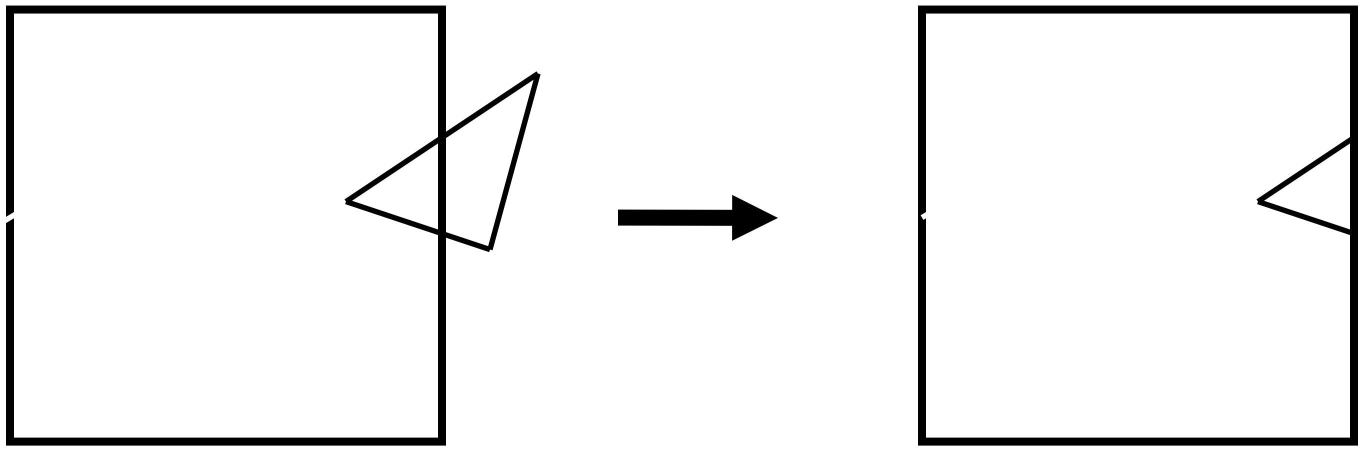
$\mathbf{M}$ is a projection matrix

GEOMETRY



Clipping and Screen Mapping

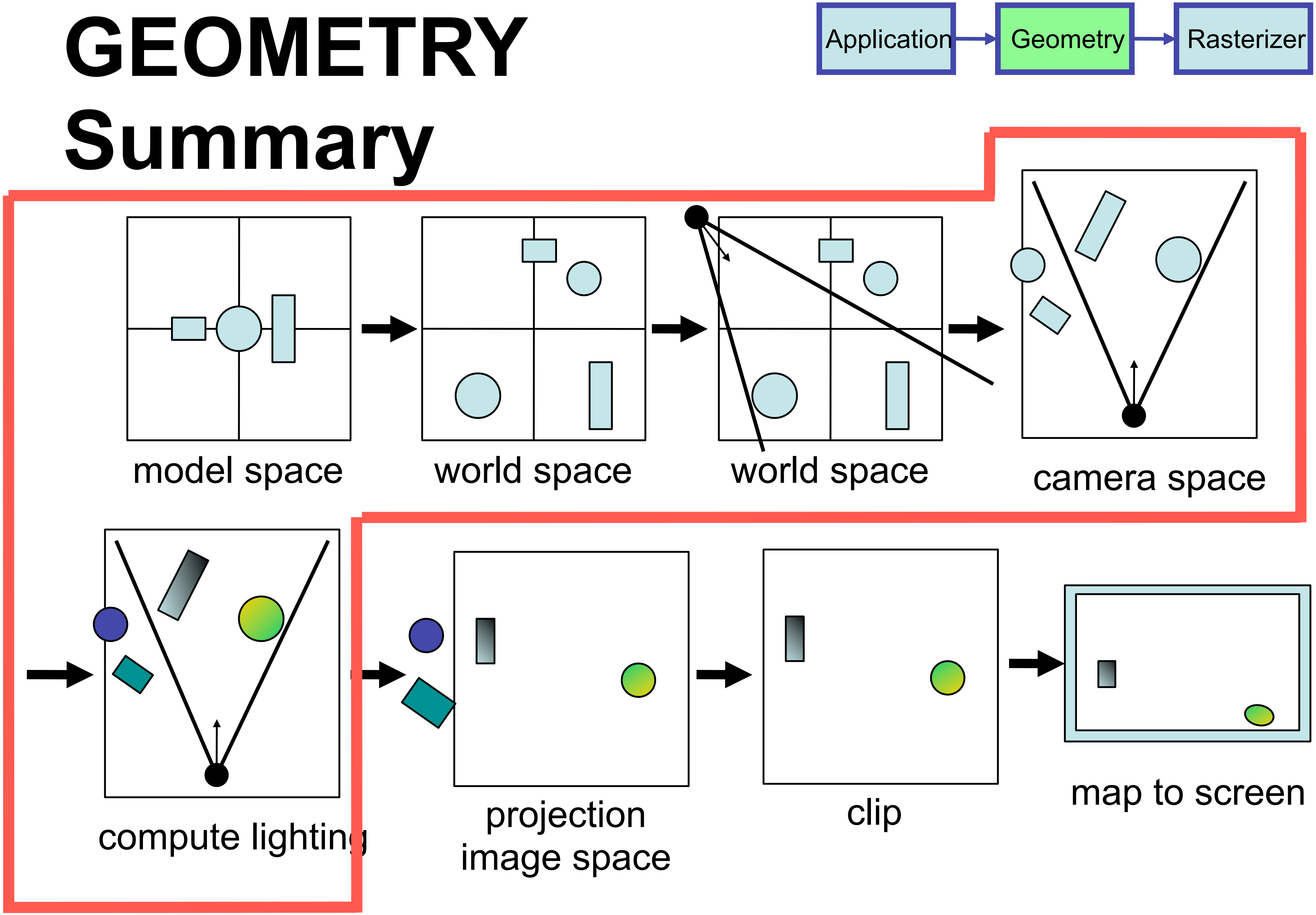
- Square (cube) after projection
- Clip primitives to square



- 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

© 2009 Tomas Akenine-Möller and Michael Doggett

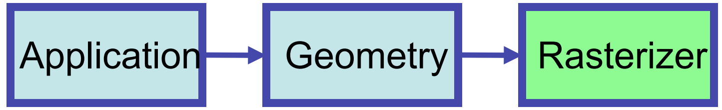
GEOMETRY Summary



Vertex Shader: does this and any other per-vertex operation

© 2009 Tomas Akenine-Möller and Michael Doggett

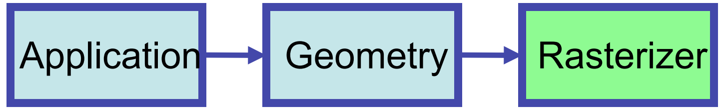
The RASTERIZER in more detail



- Scan-conversion (primitive traversal)
  - 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
- Pixel shaders (also called fragment shaders)
- And more…

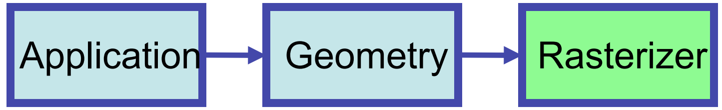
© 2009 Tomas Akenine-Möller and Michael Doggett

The RASTERIZER Scan conversion (traversal)



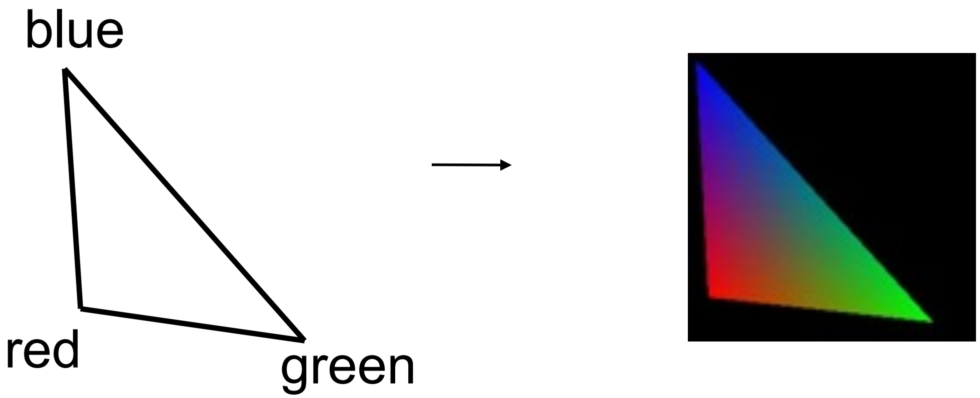
- Triangle vertices from GEOMETRY is input
- Find pixels inside the triangle
  - Or on a line, or on a point
  - We will study algorithms for this later
- Do per-pixel operations on these pixels:
  - Interpolation (lecture)
  - Texturing (lectures on how to reduce BW)
  - Z-buffering (lecture on how to compress)
  - And more…

© 2009 Tomas Akenine-Möller and Michael Doggett



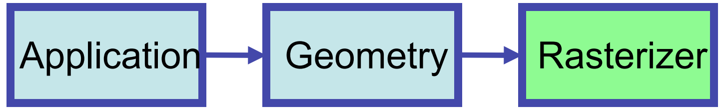
The RASTERIZER Interpolation

- Interpolate colors over the triangle
  - Called Gouraud interpolation

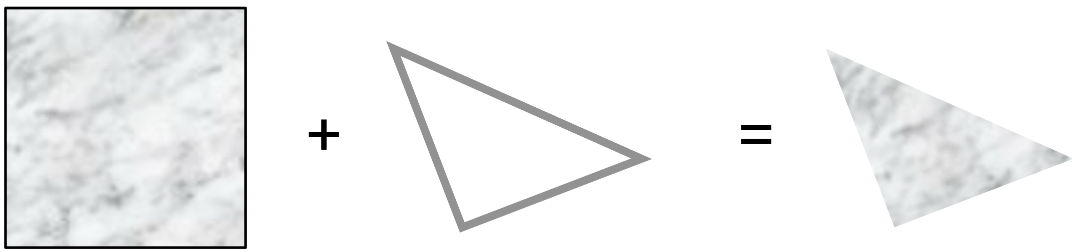


© 2009 Tomas Akenine-Möller and Michael Doggett

The RASTERIZER Texturing

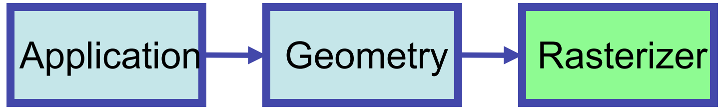


- One application of texturing is to ”glue” images onto geometrical object



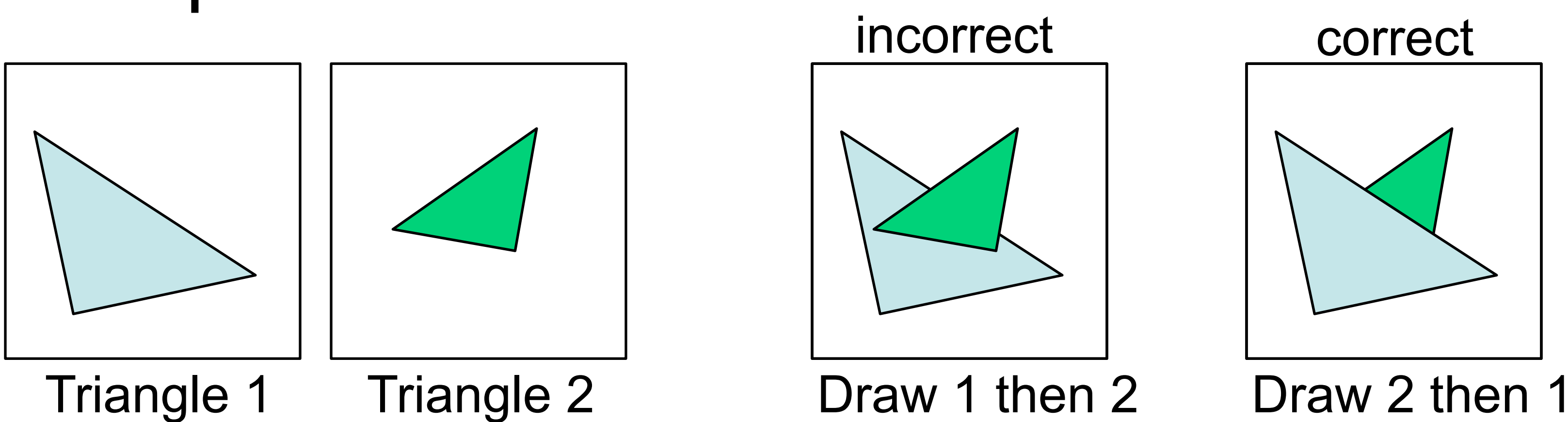
- Uses and other applications
  - More realism
  - Bump mapping
  - Pseudo reflections
  - Store lighting
  - Almost infinitely many uses

© 2009 Tomas Akenine-Möller and Michael Doggett



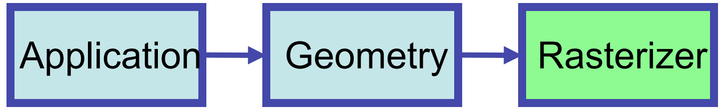
The RASTERIZER
Z-buffering

- 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



© 2009 Tomas Akenine-Möller and Michael Doggett

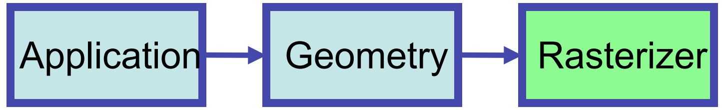
The RASTERIZER
Z-buffering



- Would be nice to avoid sorting…
- The Z-buffer (aka depth buffer) solves this
- Idea:
  - Store z (depth) at each pixel
  - When scan-converting (traversing) 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

© 2009 Tomas Akenine-Möller and Michael Doggett

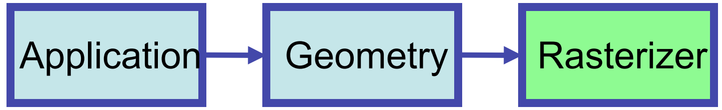
The RASTERIZER Double buffering



- 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”

© 2009 Tomas Akenine-Möller and Michael Doggett

The RASTERIZER Double buffering



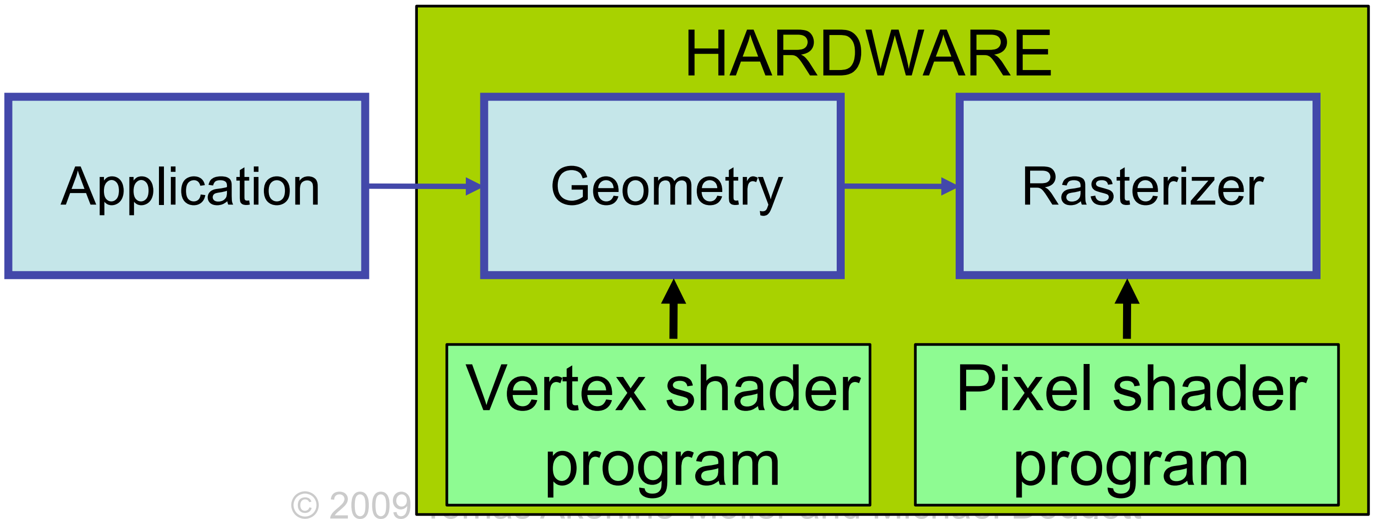
- Use two buffers: one front and one back
- The front buffer is displayed
- The back buffer is rendered to
- When new image has been created in back buffer, swap front and back

© 2009 Tomas Akenine-Möller and Michael Doggett

Shaders

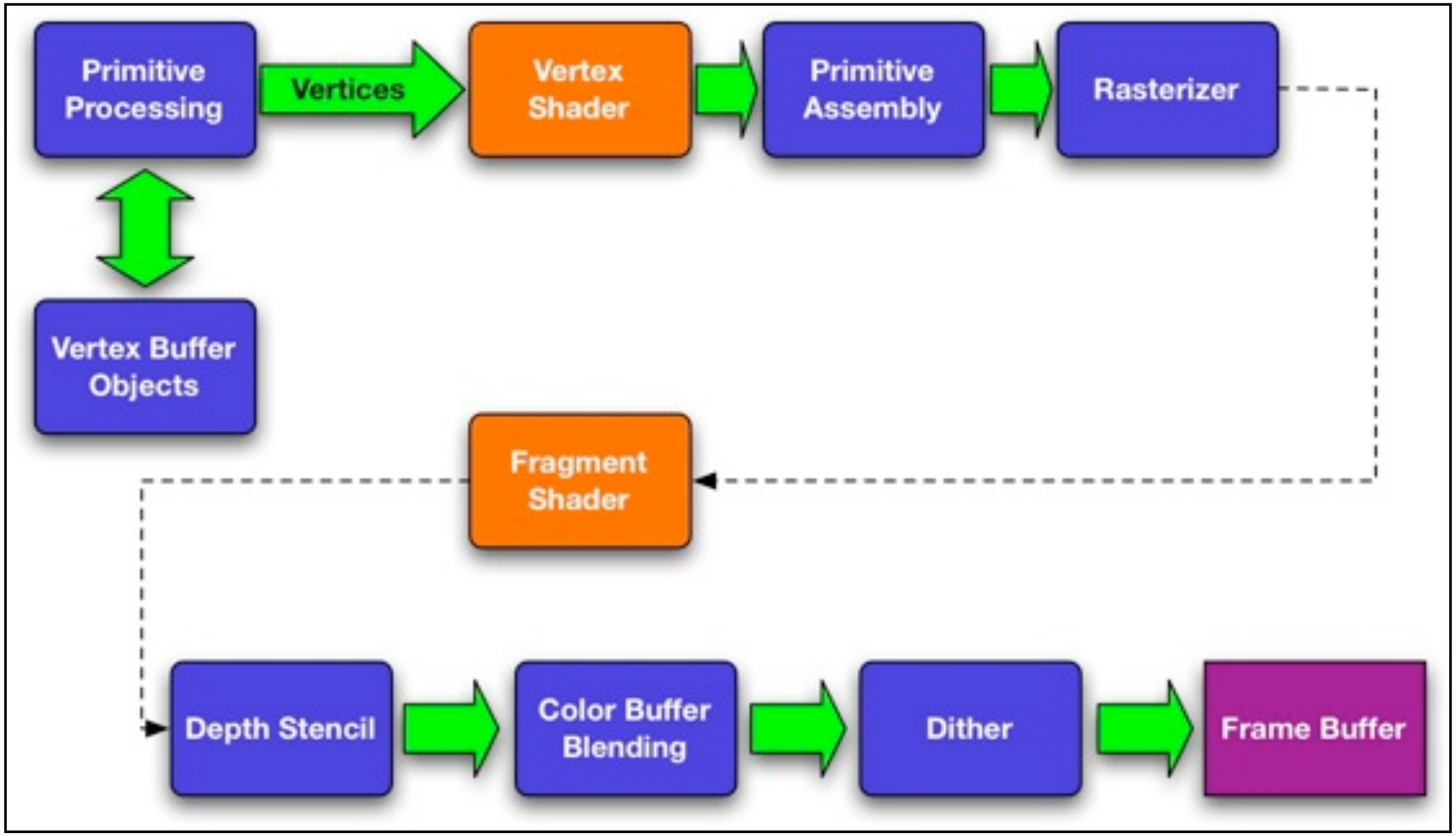
- Programmable shading has become a hot topic
  - Vertex shaders (arbitrary per-vertex ops)
  - Pixel shaders (arbitrary per-fragment ops)
  - Adds more control and much more possibilities for the programmer

Real-time screenshot from another course: Advanced Shading And Rendering, VT2, LTH



© 2009

Another pipeline diagram with shaders



© 2009 Tomas Akenine-Möller and Michael Doggett

What you should know by now

- You should have the rendering pipeline fresh in mind
- If this still feels a bit odd:
  - play with OpenGL for a while
    - Example program on web page

© 2009 Tomas Akenine-Möller and Michael Doggett

Next few lectues

- Focus on SceneGraph Framework
  - First lecture: API overview: OpenGL ES and Scene Graphs
  - Second lecture (given by Magnus Andersson)
    - Is about the first programming assignment
    - Should be available on 2009-10-29 (see course website for more details)

© 2009 Tomas Akenine-Möller and Michael Doggett

Check course website regularly

http://cs.lth.se/eda075/

The end

© 2009 Tomas Akenine-Möller and Michael Doggett