XNA@DSI

Giuseppe Maggiore
Microsoft Student Partner
University Ca' Foscari of Venice

xna | Game Studio
European Tour 2007

Microsoft

Computer Graphics

=pretty pictures

of possibly moving, possibly interactive, solid or fluid, artificial or living things for people to see on displays

Who needs computer graphics?

- Computer-Aided Design/Manufacturing
- Medical Imaging
- Simulation
- Architecture
- Electronic publishing
- Computer Animation / Film Production
- Art
- Games
- …

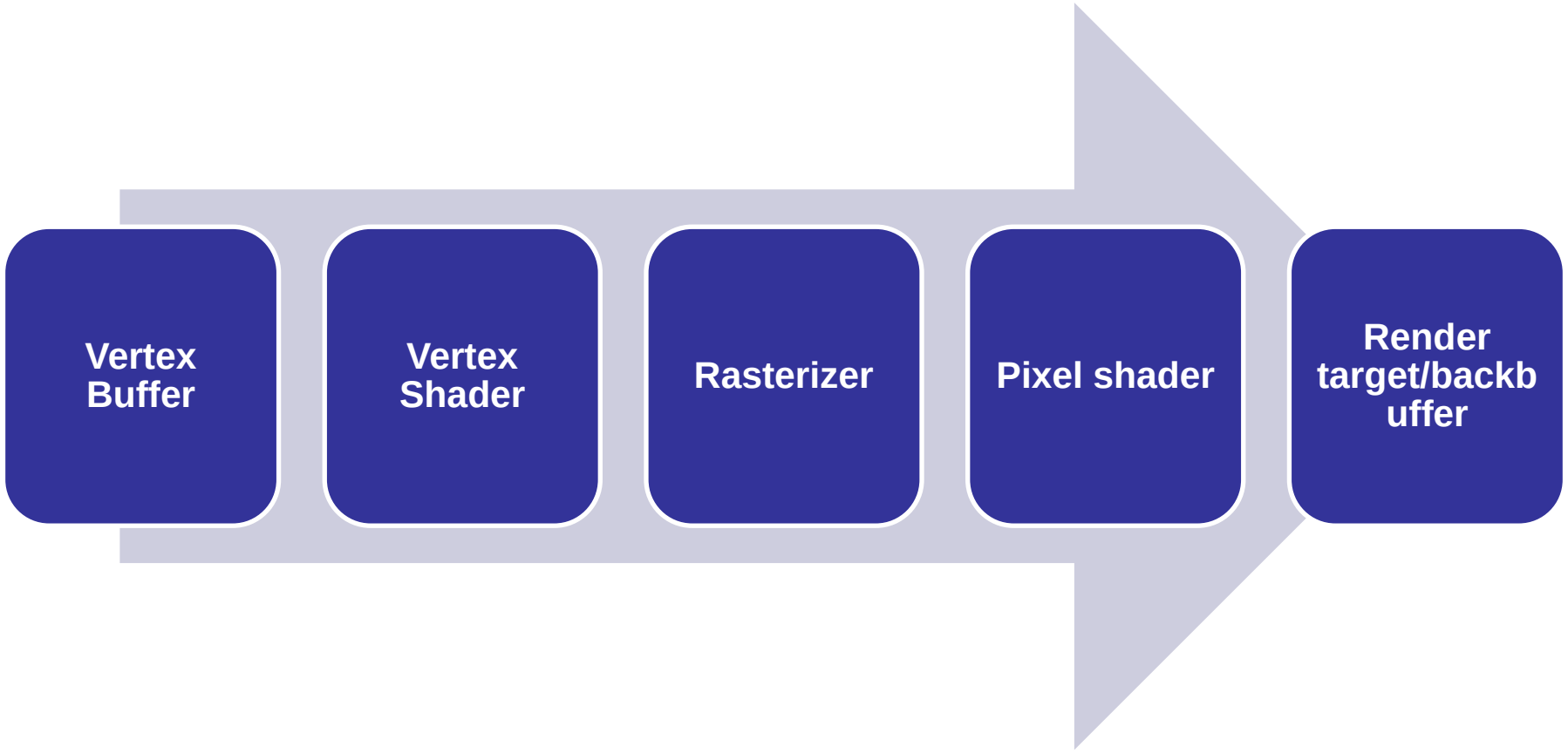
Chapter 0

Graphics Pipeline

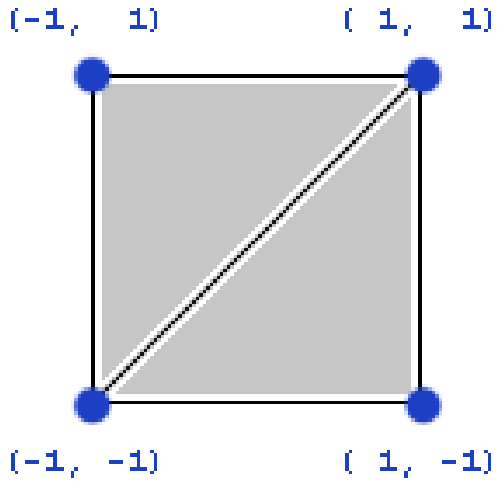
xna | Game Studio
European Tour 2007

Microsoft

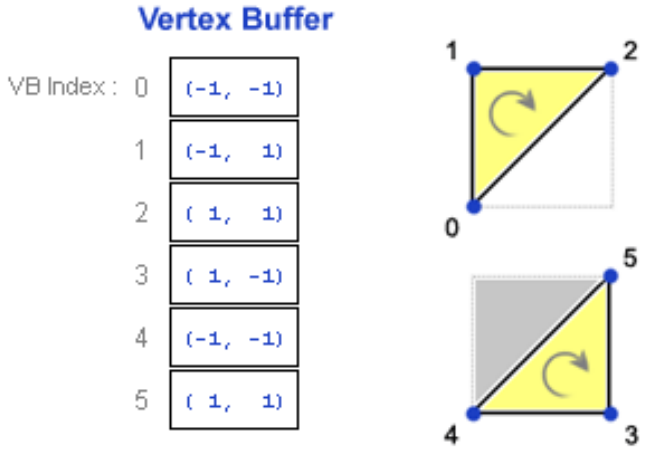
Graphics Pipeline



Vertex Buffers



Is stored as



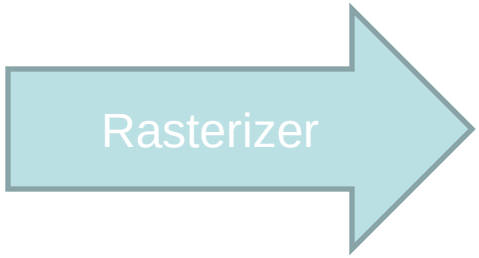
Vertex Shader

Vertex Buffer
(-1,-1,0)
(-1,+1,0)
(+1,+1,0)
(+1,-1,0)
(-1,-1,0)
(+1,+1,0)

Screen-space vertices
(+300,+400)
(+300,+600)
(+400,+600)
(+300,+400)
(+300,+400)
(+400,+600)

Rasterizer

Screen-space vertices
(+300,+400)
(+300,+600)
(+400,+600)
(+300,+400)
(+300,+400)
(+400,+600)



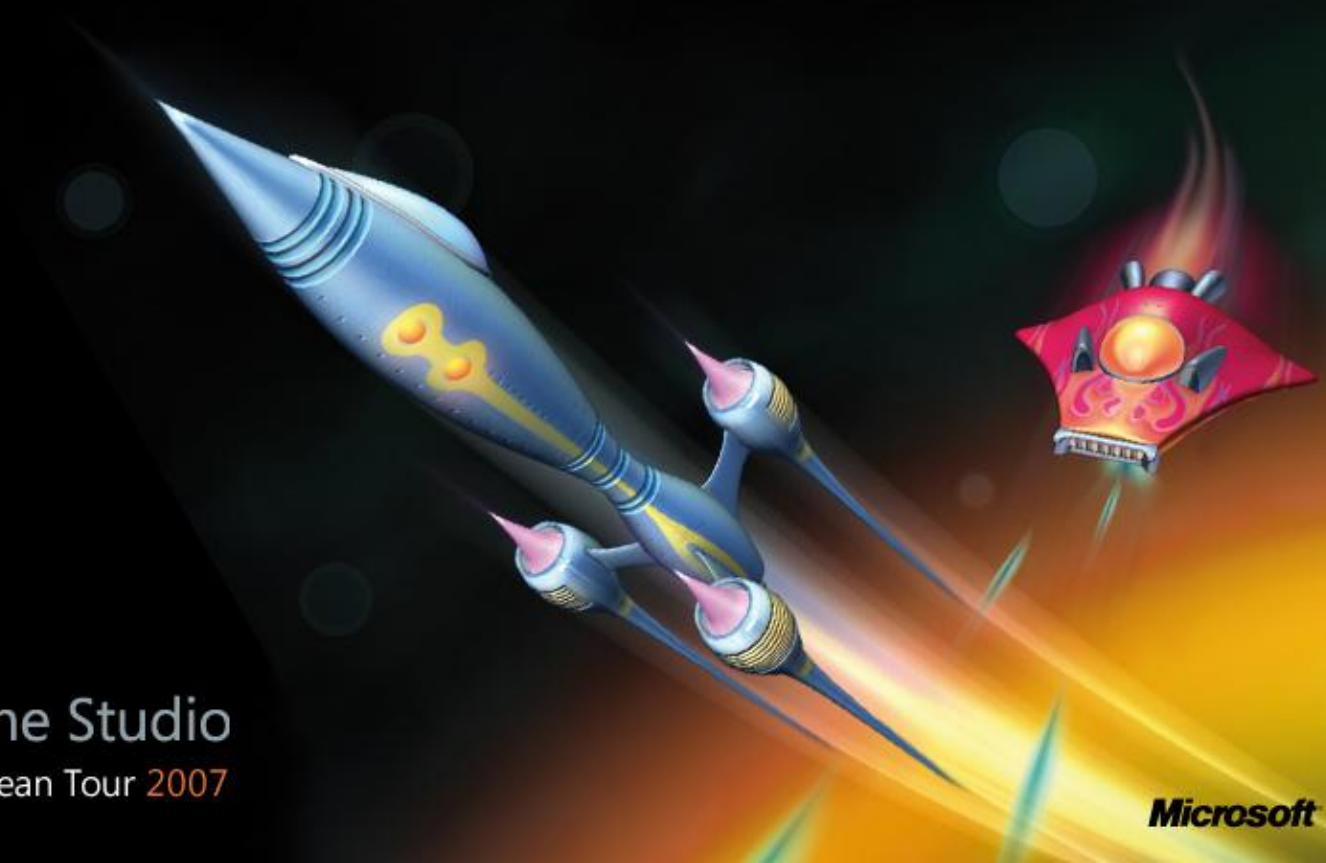
{Pixel list}

Pixel shader

{Pixel list} → Pixel shader → **{Pixel colors}**

Chapter 1

Geometry



xna | Game Studio
European Tour 2007

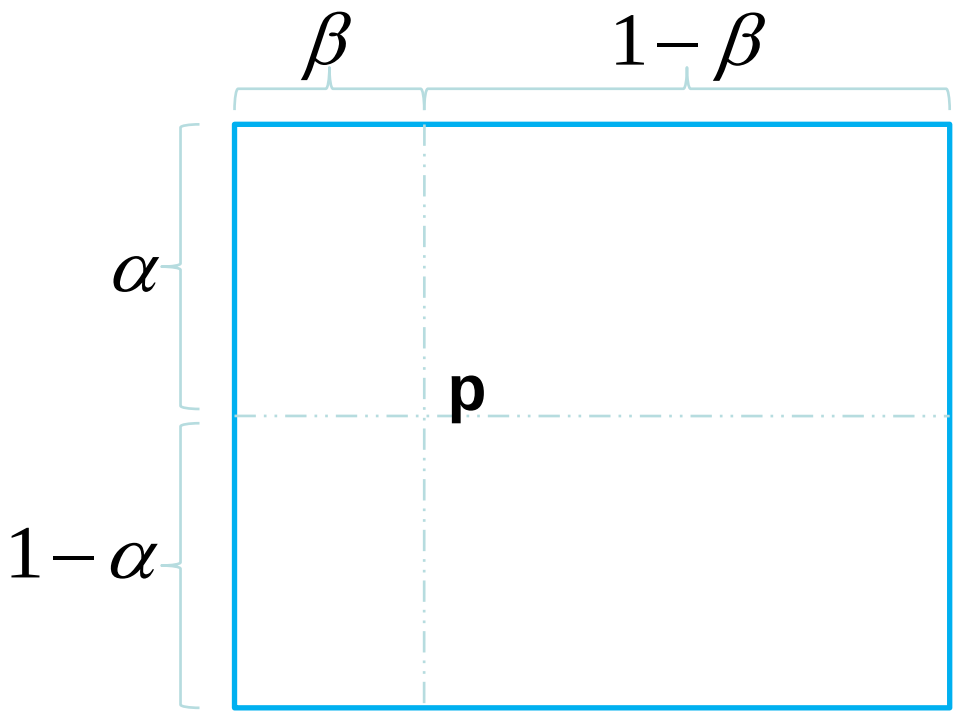
Microsoft

Geometry

- For now we will work only with vertices and triangles
- No pixel colors, no rasterizer
- Just the outline of our polygons!

Vertex buffer creation

- We will procedurally generate shapes from a quad discretely sampled at some points:

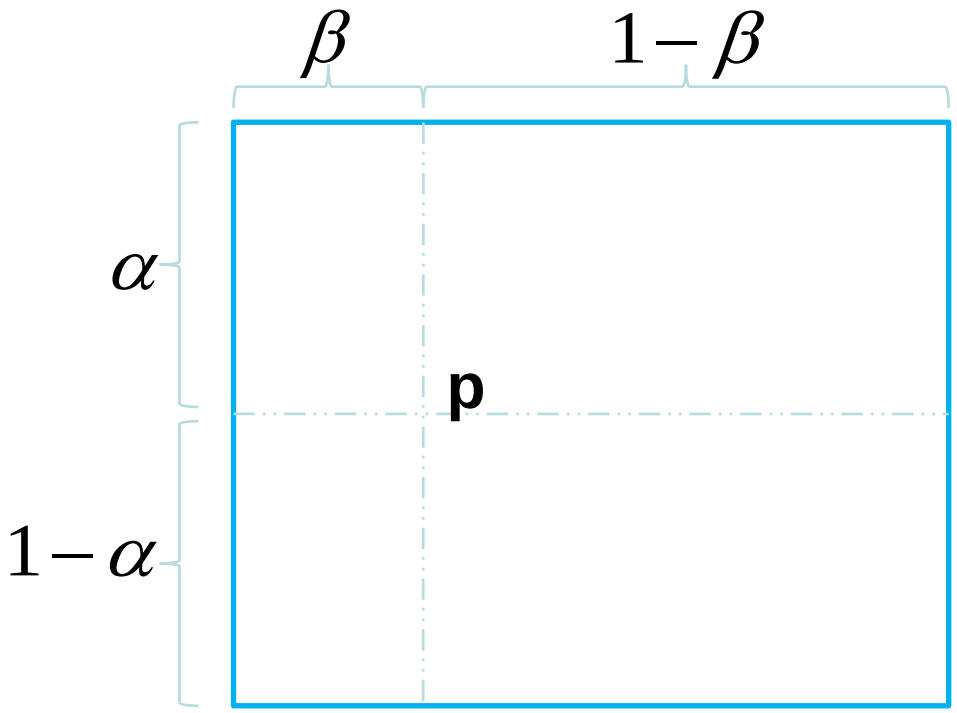


$$\boldsymbol{p \leftarrow (\beta, \alpha, 0)}$$

Demo!

Vertex buffer creation

- A more 3d-ish shape is a cylinder:



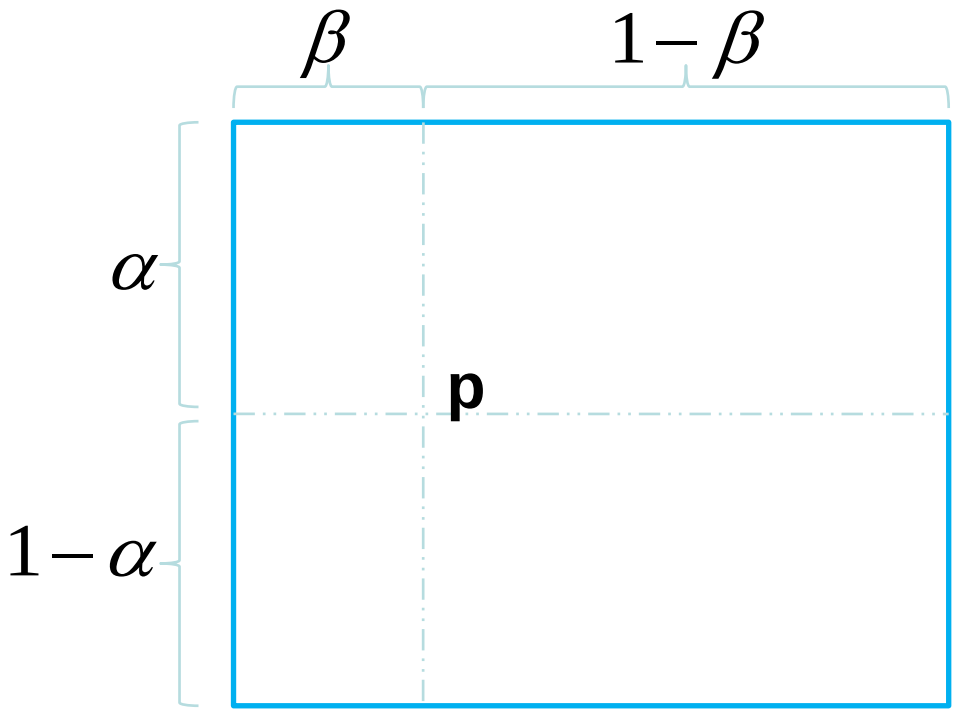
$$p \leftarrow (x, y, z)$$

$$\begin{cases} x = cos(2\pi\beta) \\ y = \alpha \\ z = sin(2\pi\beta) \end{cases}$$

Demo!

Vertex buffer creation

- Finally, a sphere:



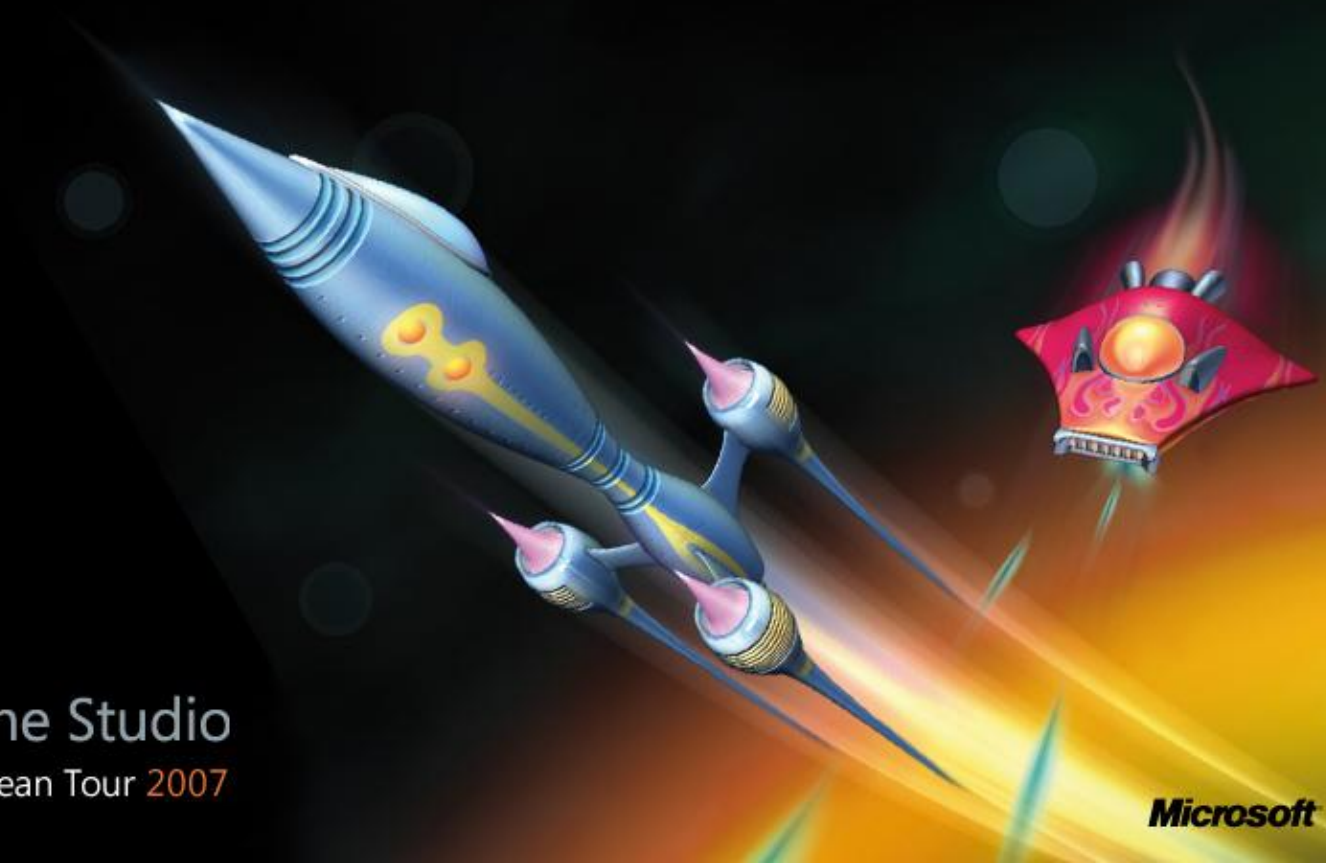
$$p \leftarrow (x, y, z)$$

$$\begin{cases} r_0 = sin(\pi\alpha) \\ x = r_0 cos(2\pi\beta) \\ y = cos(\pi\beta) \\ z = r_0 sin(2\pi\beta) \end{cases}$$

Demo!

Chapter 2

Lighting

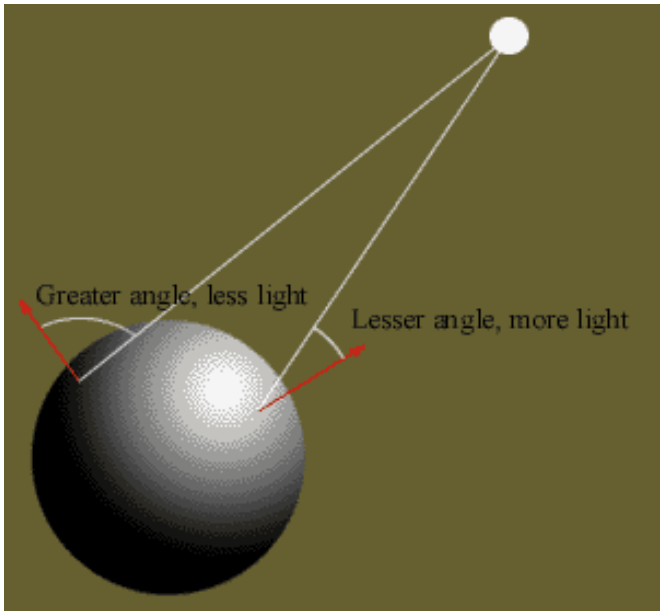


xna | Game Studio
European Tour 2007

Microsoft

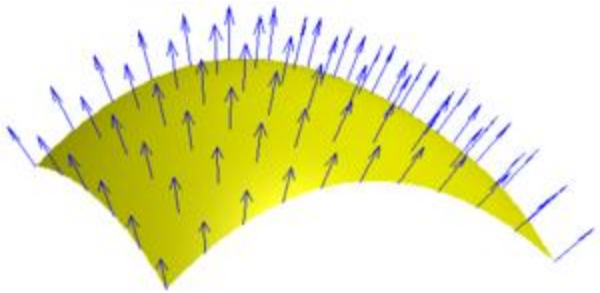
Lighting

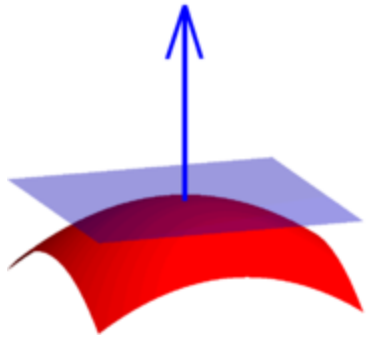
- Lighting is the computation of the contributions to each pixel color from the lights in the scene



Normal

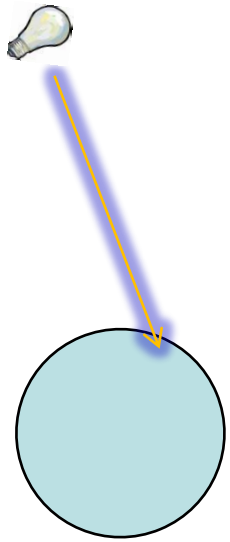
- To compute lighting we will need the normal N of the surface at each vertex
- This becomes an additional input for our vertex shader





Light direction

- The light direction L is defined as the direction from the light to the vertex



Lambert's cosine law

- the radiant intensity observed from a **Lambertian** (ideal) surface is directly proportional to the cosine of the angle θ between the light direction and the surface normal

$$light_{Lambert} \simeq \cos(L, N)$$
$$= L \cdot N$$

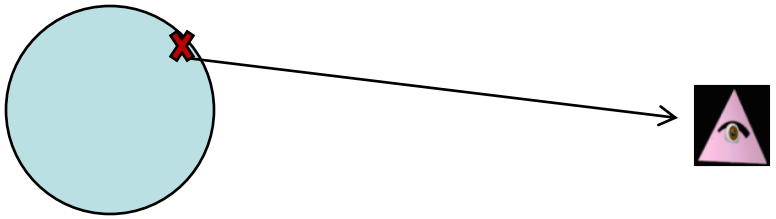
Demo!

Phong Shading

- To add specular highlights to shiny objects, we use the Phong lighting model

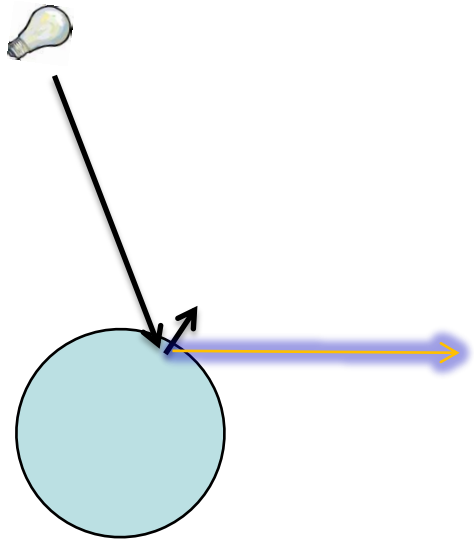
Phong Shading

- V_D is the vector that goes from the vertex to the observer



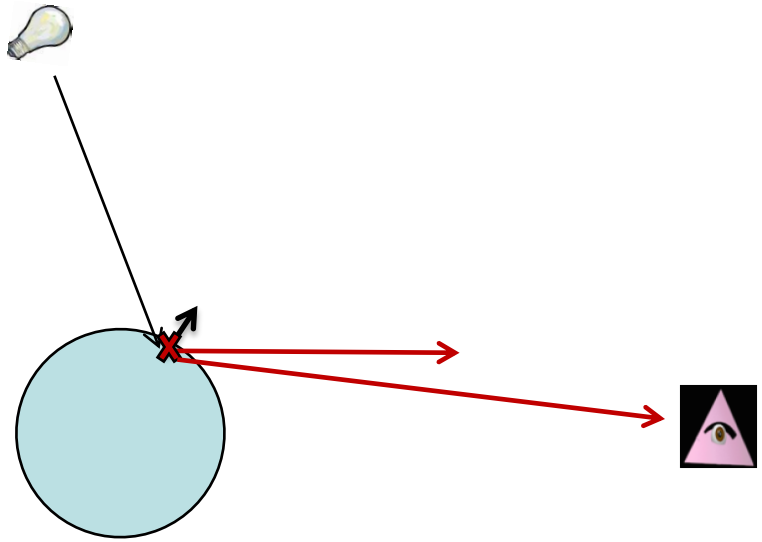
Phong Shading

- L_R is the light reflected by the surface around the vertex



Phong Shading

- The amount of reflection depends on the angle between the two vectors L_R and V_D



Phong Shading

$$light_{Phong} = \sigma(L_R \cdot V_D)^{\rho}$$

- Where sigma and rho come from the material properties

Demo!

Chapter 3

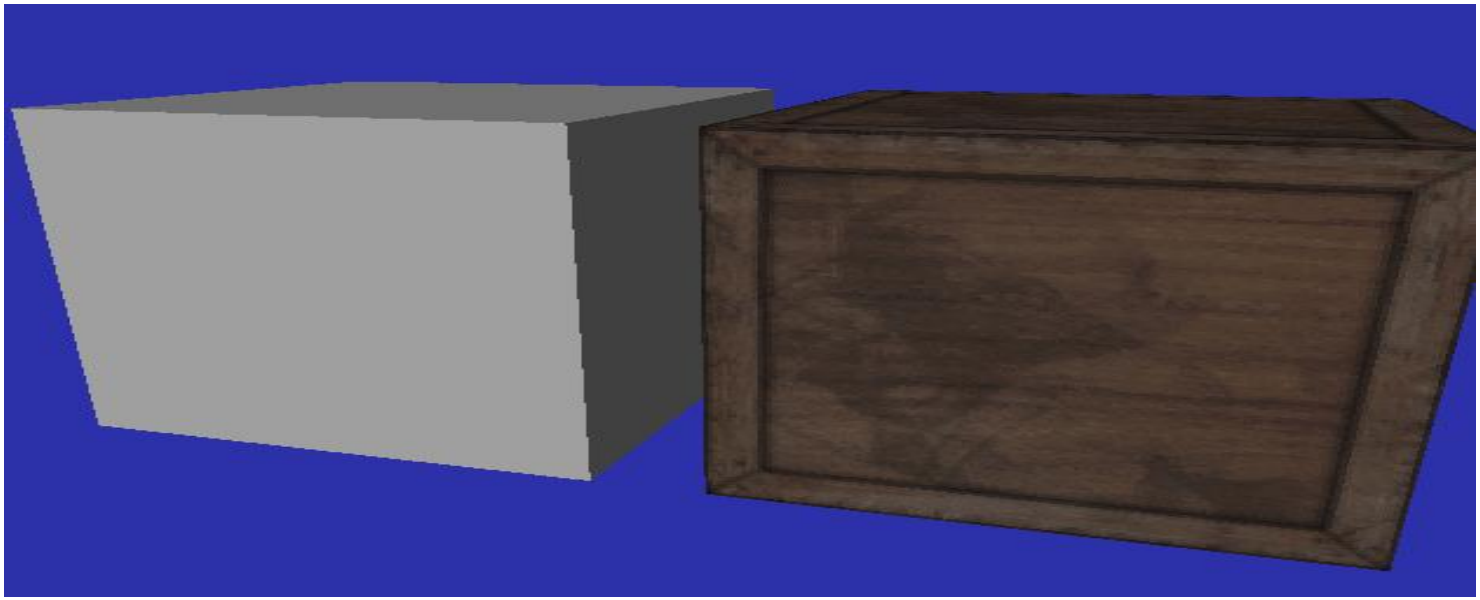
Textures

xna | Game Studio
European Tour 2007

Microsoft

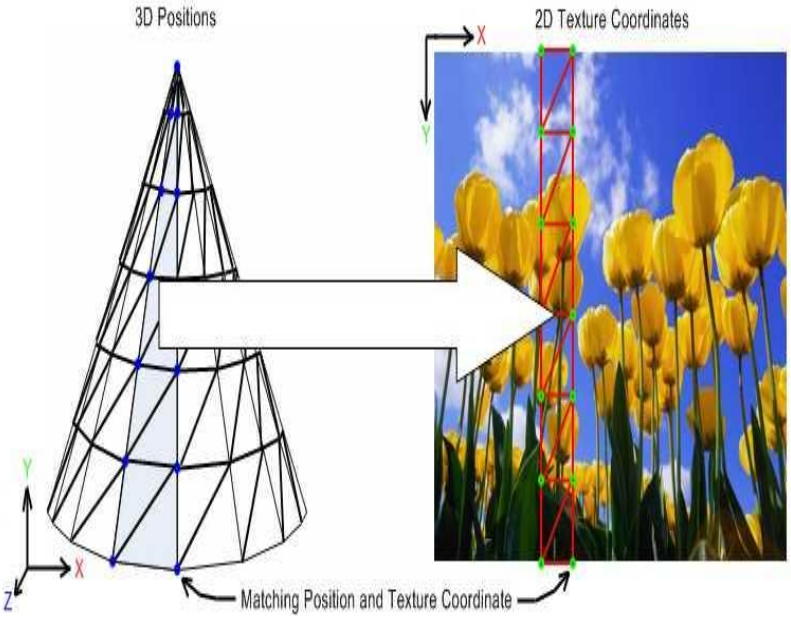
Textures

- To give a more detailed look to our polygons we smear an image over their surface



Texture coordinates

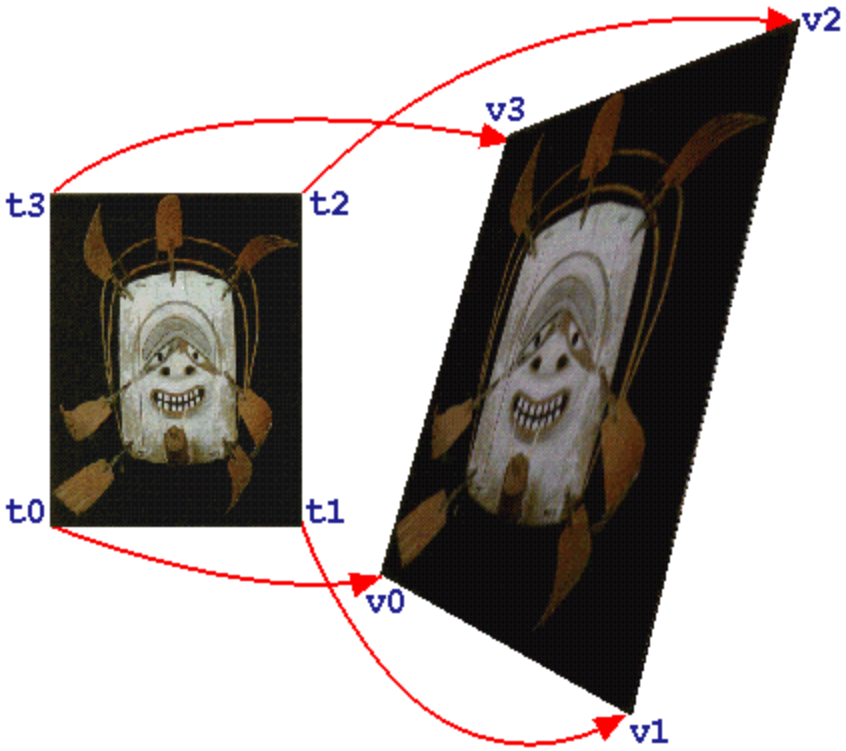
- To texture a polygon we use texture coordinates
- Texture coordinates map the vertices of our models to the 2D plane of the texture



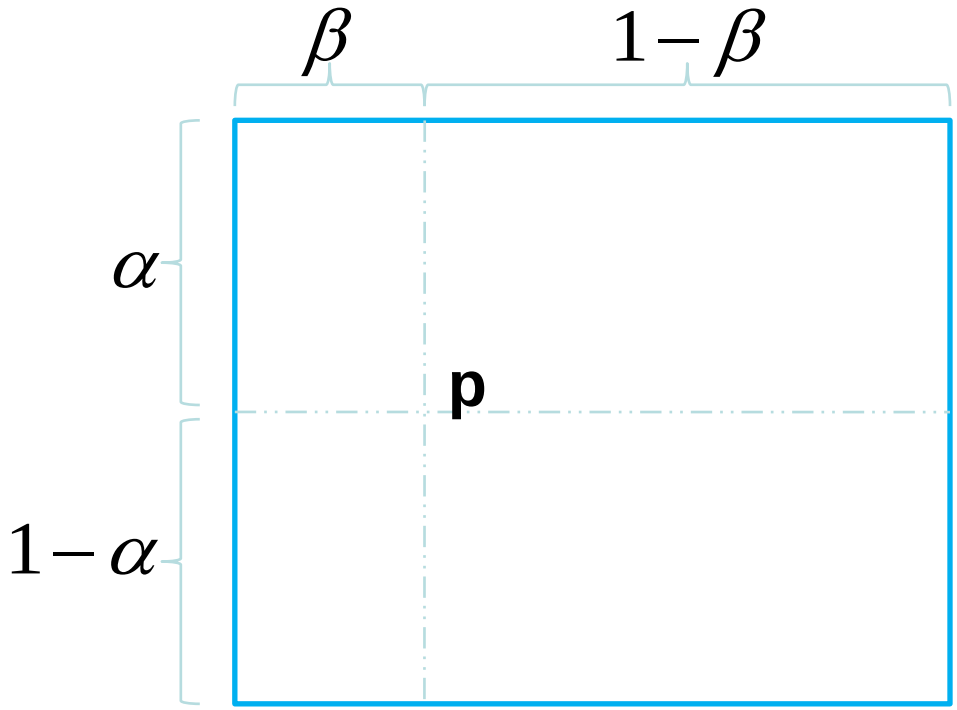
$$\psi: (x, y, z) \rightarrow (u, v)$$

$$(u, v) \in [0,1]^2$$

Texture coordinates



Quad texture coordinates



$$\psi(p) = (u, v)$$

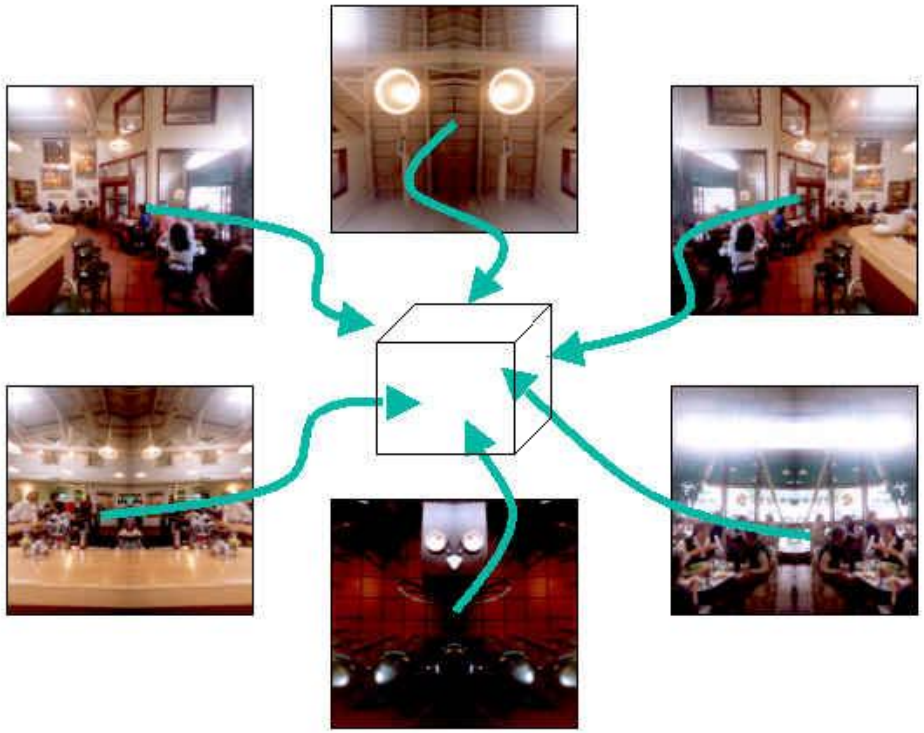
$$\begin{cases} u = \beta \\ v = 1 - \alpha \end{cases}$$

- UV are also called barycentric coordinates
- Cylinder and sphere are based on a folded quad, so they require no special treatment

Demo!

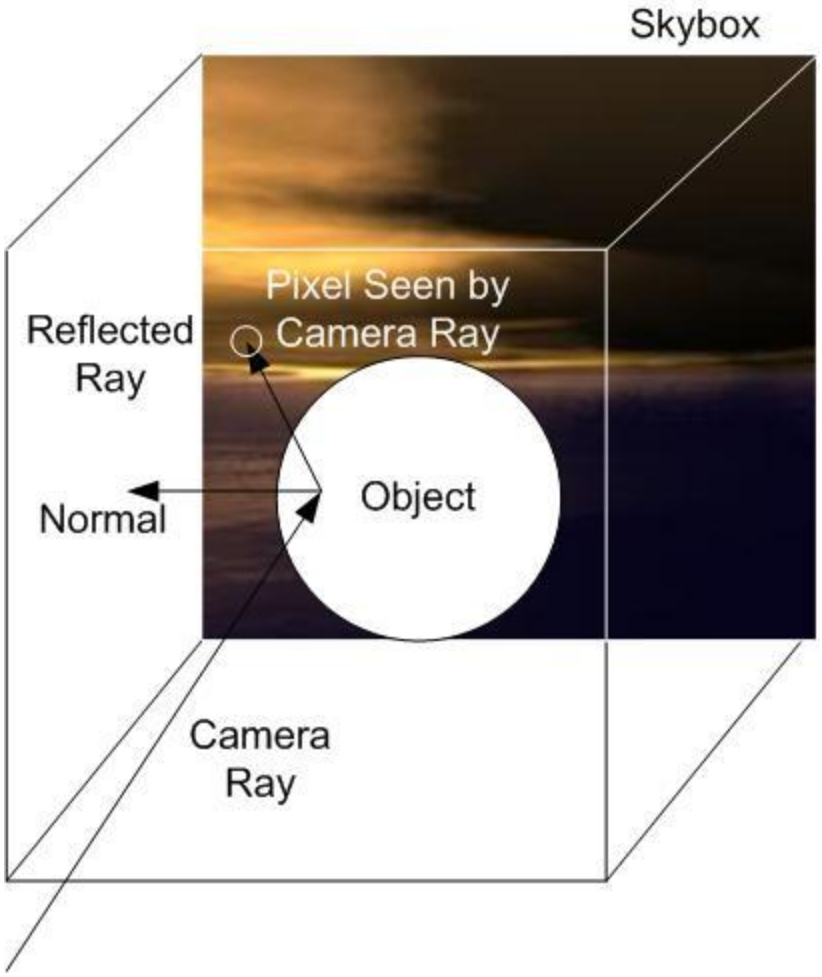
Non planar textures - example

- Textures can also have multiple faces
- Cube textures represent six different (orthogonal) views of the same scene



Cube textures

- This kind of texture is great for simulating surfaces that reflect the surrounding environment



Cube textures

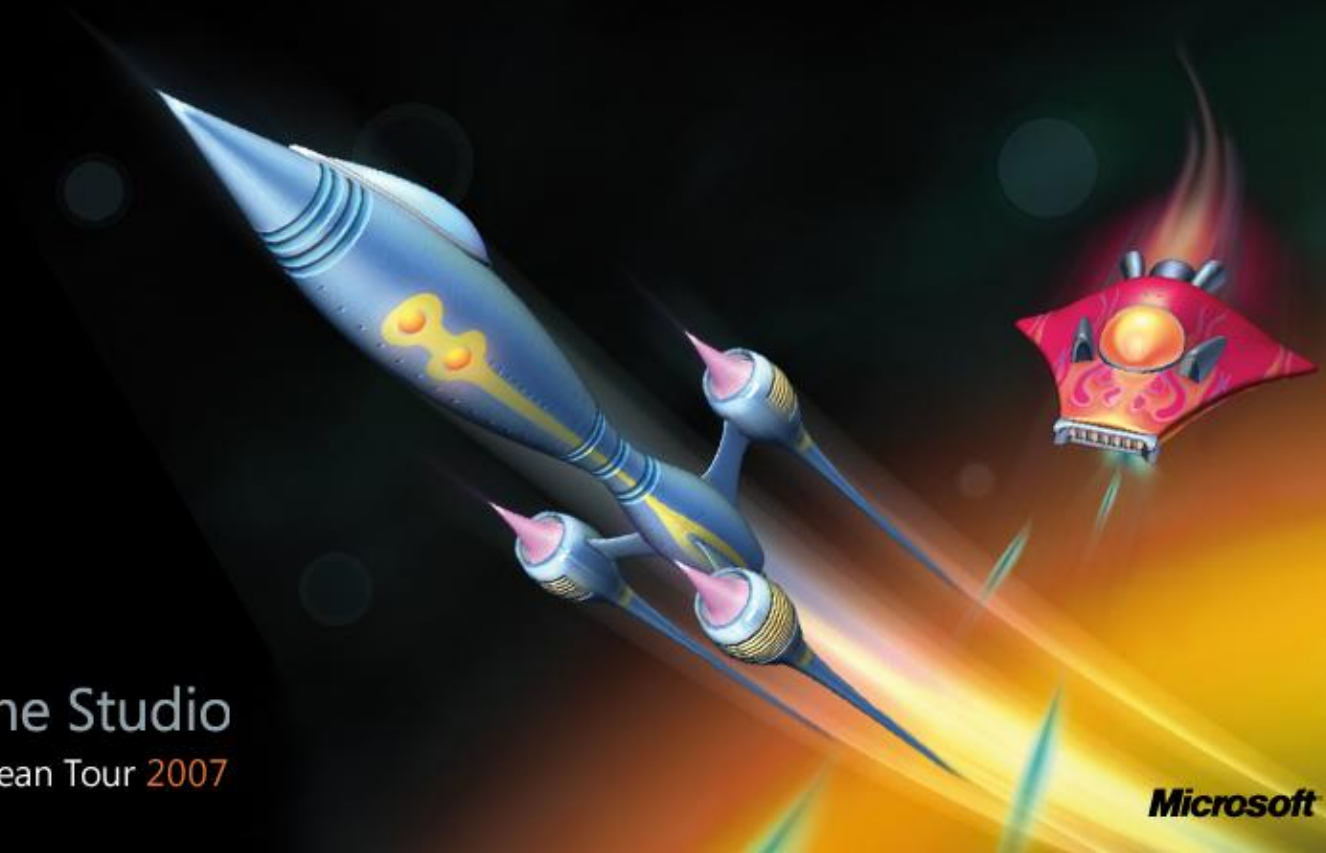
- A cube texture is sampled using 3d coordinates (no UV) that represent the direction from the center of the cube from which to take the color

$$(u, v, w) = reflect(V_D, N)$$

Demo!

Chapter 4

Simple game architecture



xna | Game Studio
European Tour 2007

Microsoft

Simple game

- To put everything together, we build a very simple sci-fi war game
- The graphics are covered (planets are spheres and quads are the units)
- Gameplay is based on:
  - **Game code structure**
  - **Data loading**
  - **Input**
  - **User interface**

Game architecture

- Game code structure
  - **A game is (almost) like an OS**
  - `while(true) do Update(); Draw();`
  - **GameComponents are anything with an update and a draw function**
  - **Input management → game component (with null draw)**
  - **User Interface → game component (with almost update)**
- Data loading
  - **XNA moves the preprocessing of all assets…**
  - **…at compile time!**
  - **At runtime we simply load our data already preprocessed and prepared for use**
  - **Think from jpeg to color matrix**
  - **Storage space versus speed (games always favor speed)**

Input

- XNA can be queried for a current snapshot of one of the following controllers:
  - **Mouse** (`Mouse.GetState()`)
  - **Keyboard** (`Keyboard.GetState()`)
  - **XBox 360 controller** (`GamePad.GetState(PlayerIndex)`)
- This snapshot should be updated at every frame (Update() function)
- Often previous states are needed (hysteresis)

User interface

- XNA provides us with a very powerful, efficient and easy to use class: `SpriteBatch`
  - **Double buffering for high performance**
  - **Simple interface**
  - **Can optionally use custom shaders**

Demo!

GAME OVER!

xna | Game Studio
European Tour 2007

Microsoft