CSE 167: Introduction to Computer Graphics Lecture #12: Environment Mapping

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2016

Announcements

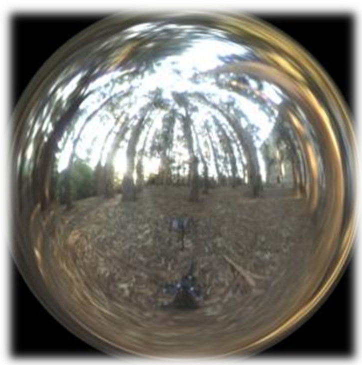
- This Thursday: Midterm 2
- No grading Friday (Veterans Day)
- Late grading project 3 Thursday 3:30-4:30pm
  - or next week during office hours
  - Code submission on Ted by Friday 2pm required

More Realistic Illumination

- In the real world:
  At each point in scene light arrives from all directions
  - Not just from a few point light sources
  - → Global Illumination is a solution, but computationally expensive
- Environment Maps
  - Store "omni-directional" illumination as images
  - Each pixel corresponds to light from a certain direction
  - Sky boxes make for great environment maps

Capturing Environment Maps

- Environment map = surround panoramic image
- Creating 360 degrees panoramic images:
  - 360 degree camera
  - "light probe" image: take picture of mirror ball (e.g., silver Christmas ornament)



Light Probes by Paul Debevec
http://www.debevec.org/Probes/

Environment Maps as Light Sources

Simplifying Assumption

- Assume light captured by environment map is emitted from infinitely far away
- Environment map consists of directional light sources
  - Value of environment map is defined for each **direction**, independent of position in scene
- Approach uses same environment map at each point in scene
  → Approximation!

Applications for Environment Maps

- Use environment map as "light source"

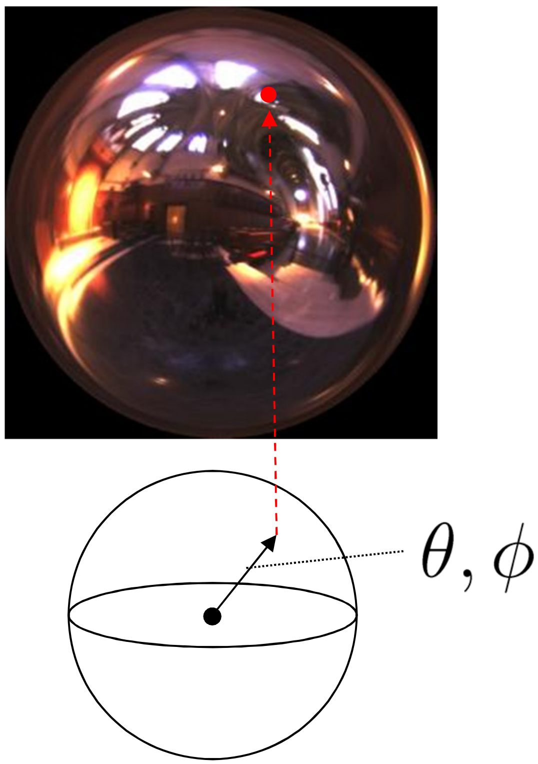
Global illumination with pre-computed radiance transfer [Sloan et al. 2002]

Reflection mapping [Georg-Simon Ohm University of Applied Sciences]

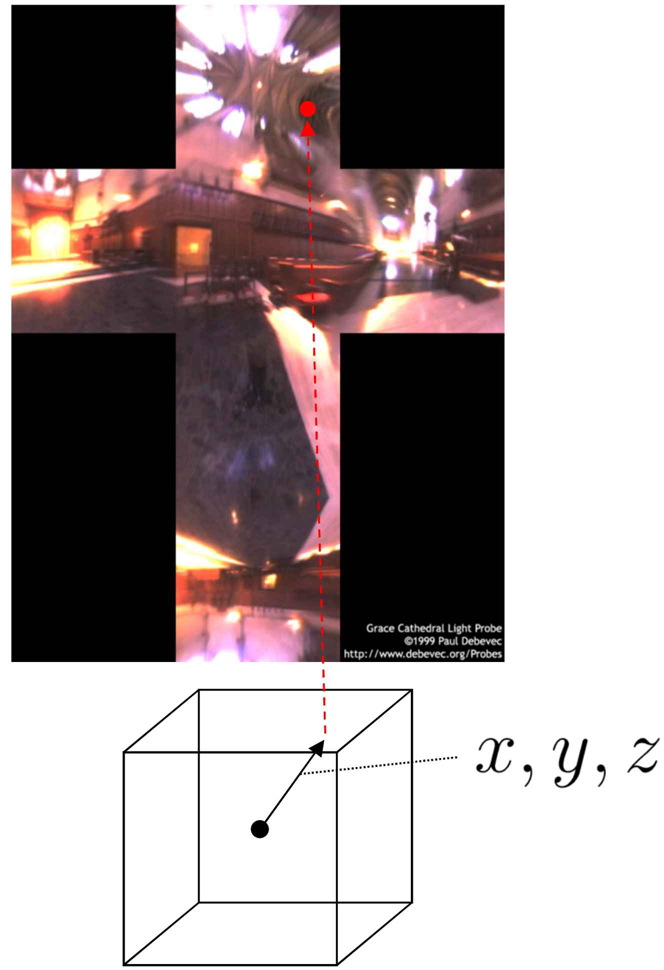
UCSD

Cubic Environment Maps

- Store incident light on six faces of a cube instead of on sphere



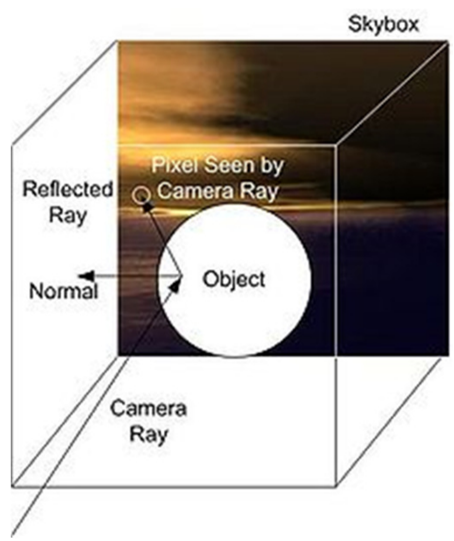
Spherical map



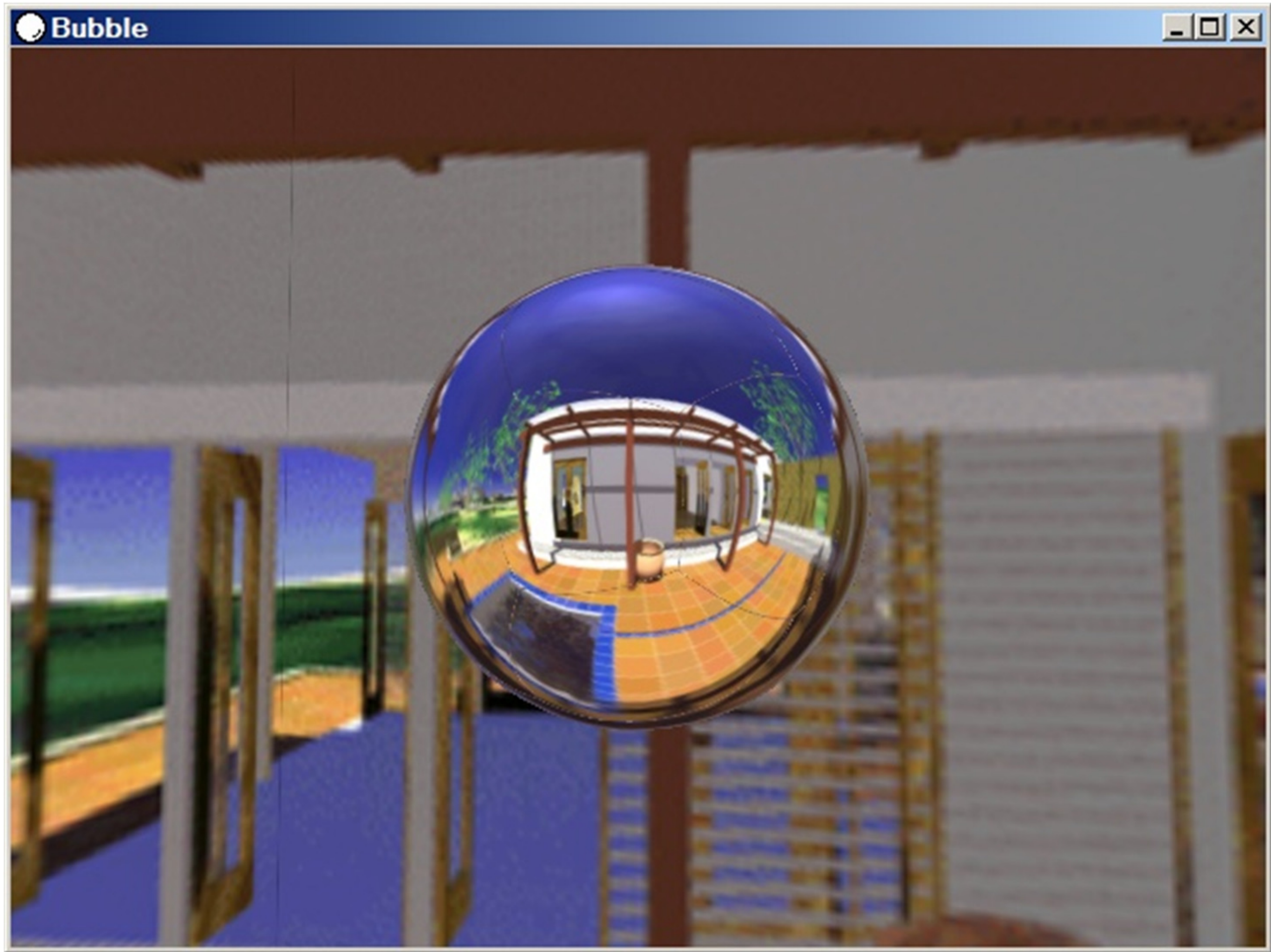
Cube map

Cubic vs. Spherical Maps

- Advantages of cube maps:
  - More even texel sample density causes less distortion, allowing for lower resolution maps
  - Easier to dynamically generate cube maps for real-time simulated reflections



Bubble Demo



http://download.nvidia.com/downloads/nZone/demos/nvidia/Bubble.zip

UCSD

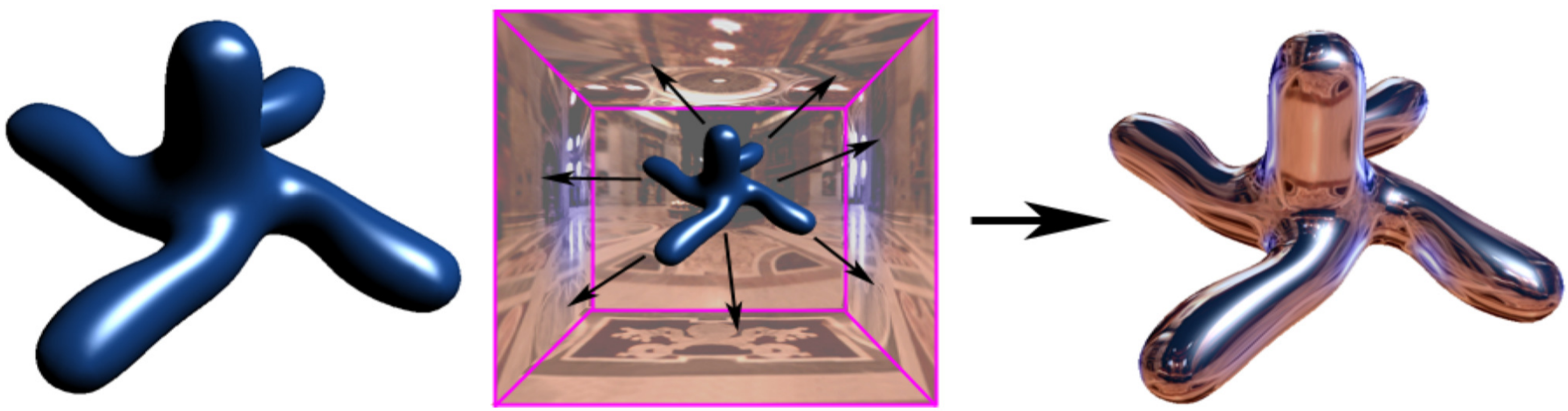
Cubic Environment Maps

Cube map look-up

- Given: light direction (x,y,z)
- Largest coordinate component determines cube map face
- Dividing by magnitude of largest component yields coordinates within face
- In GLSL:
  - Use (x,y,z) direction as texture coordinates to `samplerCube`

Reflection Mapping

- Simulates mirror reflection
- Computes reflection vector at each pixel
- Use reflection vector to look up cube map
- Rendering cube map itself is optional (application dependent)



Reflection mapping

Reflection Mapping in GLSL

Application Setup

- Load and bind a cube environment map

```
glBindTexture(GL_TEXTURE_CUBE_MAP, …);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X,…);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X,…);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y,…);
…
glEnable(GL_TEXTURE_CUBE_MAP);
```

Reflection Mapping in GLSL

Vertex shader

- Compute viewing direction
- Reflection direction
  - Use `reflect` function
- Pass reflection direction to fragment shader

Fragment shader

- Look up cube map using interpolated reflection direction

```
varying float3 refl;
uniform samplerCube envMap;
textureCube(envMap, refl);
```

Environment Maps as Light Sources

- Covered so far: shading of a specular surface

→ How do you compute shading of a diffuse surface?

Diffuse Irradiace Environment Map

- Given a scene with k directional lights, light directions $d_1..d_k$ and intensities $i_1..i_k$, illuminating a diffuse surface with normal n and color c
- Pixel intensity B is computed as: $B = c \sum_{j=1..k} \max(0, d_j \cdot n) i_j$
- Cost of computing B proportional to number of texels in environment map!
- → Precomputation of diffuse reflection
- Observations:
  - All surfaces with normal direction n will return the same value for the sum
  - The sum is dependent on just the lights in the scene and the surface normal
- Precompute sum for any normal n and store result in a second environment map, indexed by surface normal
- Second environment map is called *diffuse irradiance environment map*
- Allows to illuminate objects with arbitrarily complex lighting environments with single texture lookup

Diffuse Irradiace Environment Map

- Two cubic environment maps:
  - Reflection map
  - Diffuse map

- Diffuse shading vs. shading w/diffuse map

Image source: http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter10.html

Bilinear Patches

Curved Surfaces

Curves

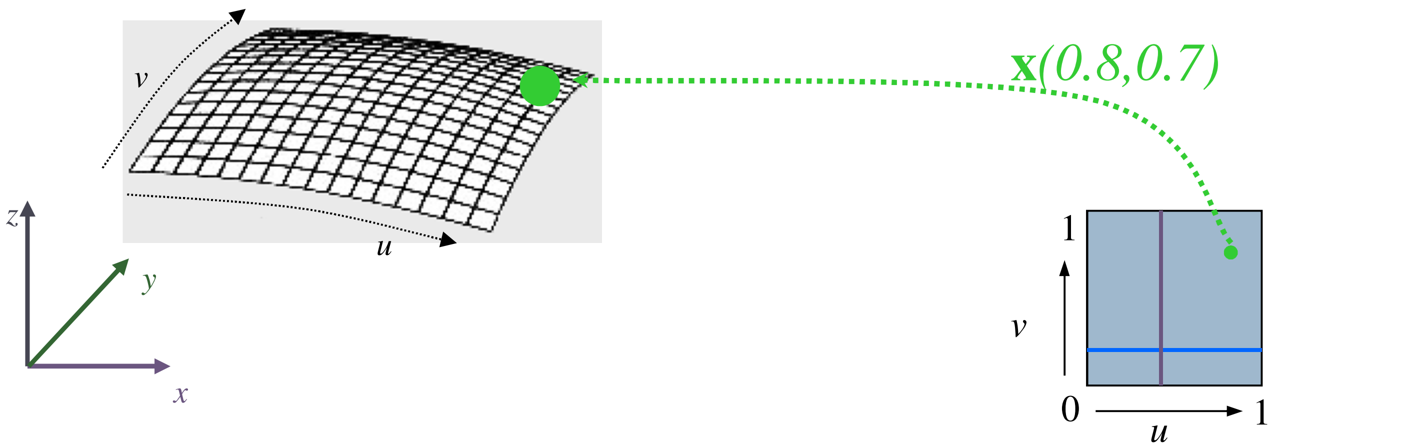
- Described by a 1D series of control points
- A function $\mathbf{x}(t)$
- Segments joined together to form a longer curve

Surfaces

- Described by a 2D mesh of control points
- Parameters have two dimensions (two dimensional parameter domain)
- A function $\mathbf{x}(u,v)$
- Patches joined together to form a bigger surface

Parametric Surface Patch

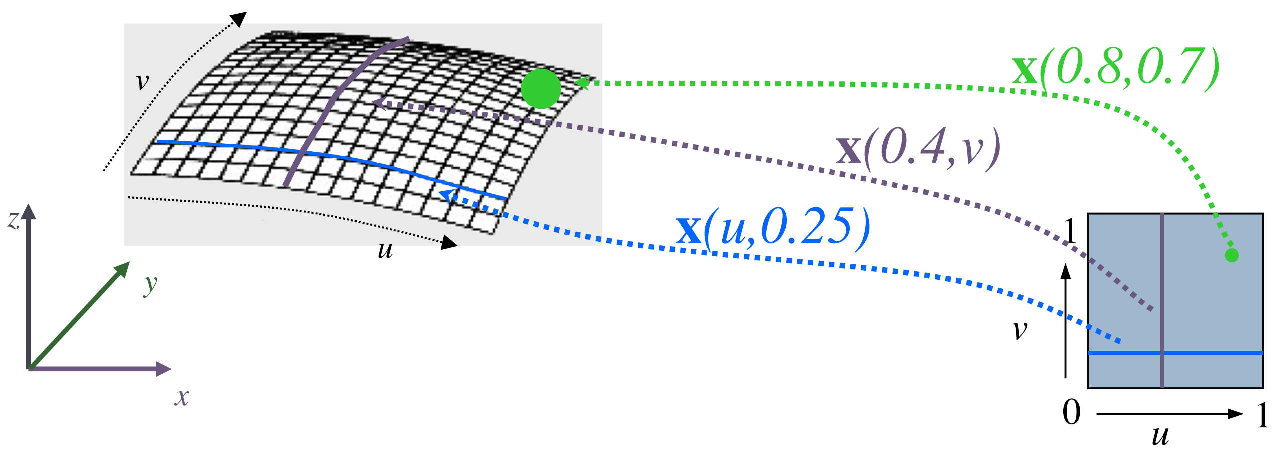
- $\mathbf{x}(u,v)$ describes a point in space for any given (u,v) pair
  - u,v each range from 0 to 1



2D parameter domain

Parametric Surface Patch

- $\mathbf{x}(u,v)$ describes a point in space for any given (u,v) pair
  - u,v each range from 0 to 1

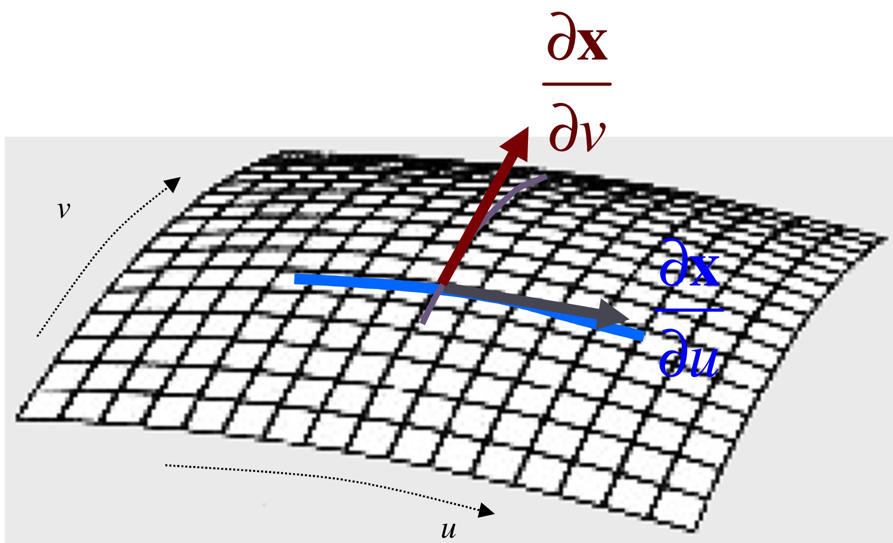


2D parameter domain

- Parametric curves
  - For fixed u_0, have a v curve $\mathbf{x}(u_0,v)$
  - For fixed v_0, have a u curve $\mathbf{x}(u,v_0)$
  - For any point on the surface, there are a pair of parametric curves through that point

Tangents

- The tangent to a parametric curve is also tangent to the surface
- For any point on the surface, there are a pair of (parametric) tangent vectors
- Note: these vectors are not necessarily perpendicular to each other



UCSD

Tangents

- Notation:
  - The tangent along a u curve, AKA the tangent in the u direction, is written as:

$$\frac{\partial \mathbf{x}}{\partial u}(u,v) \text{ or } \tfrac{\partial}{\partial u}\mathbf{x}(u,v) \text{ or } \mathbf{x}_u(u,v)$$

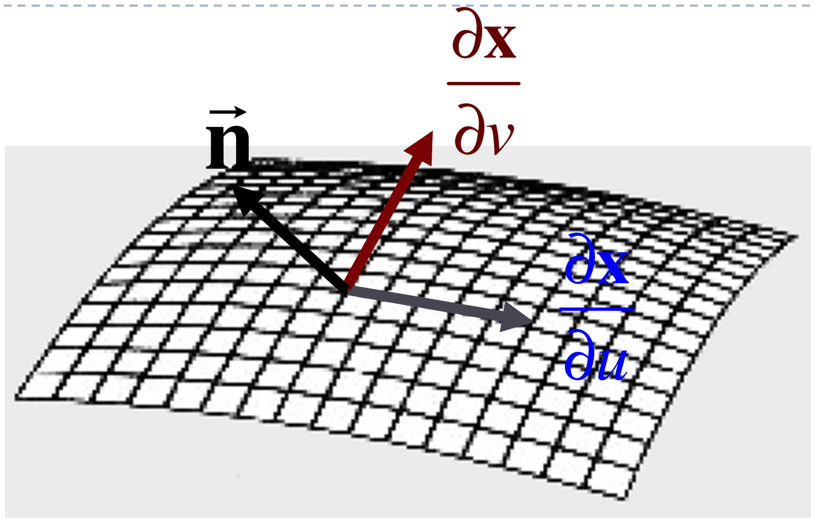
  - The tangent along a v curve, AKA the tangent in the v direction, is written as:

$$\frac{\partial \mathbf{x}}{\partial v}(u,v) \text{ or } \tfrac{\partial}{\partial v}\mathbf{x}(u,v) \text{ or } \mathbf{x}_v(u,v)$$

- Note that each of these is a vector-valued function:
  - At each point $\mathbf{x}(u,v)$ on the surface, we have tangent vectors $\tfrac{\partial}{\partial u}\mathbf{x}(u,v)$ and $\tfrac{\partial}{\partial v}\mathbf{x}(u,v)$

Surface Normal

- Normal is cross product of the two tangent vectors
- Order matters!



$$\vec{\mathbf{n}}(u,v) = \frac{\partial \mathbf{x}}{\partial u}(u,v) \times \frac{\partial \mathbf{x}}{\partial v}(u,v)$$

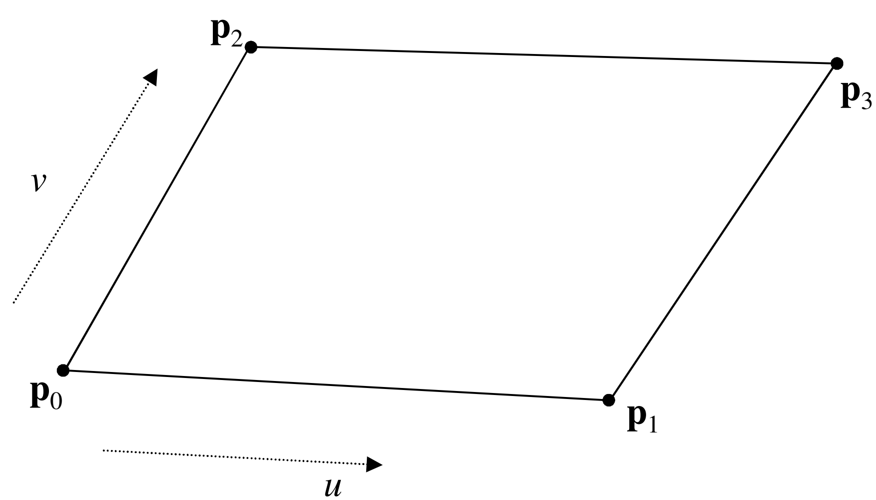
Typically we are interested in the unit normal, so we need to normalize

$$\vec{\mathbf{n}}^*(u,v) = \frac{\partial \mathbf{x}}{\partial u}(u,v) \times \frac{\partial \mathbf{x}}{\partial v}(u,v)$$

$$\vec{\mathbf{n}}(u,v) = \frac{\vec{\mathbf{n}}^*(u,v)}{\left|\vec{\mathbf{n}}^*(u,v)\right|}$$

Bilinear Patch

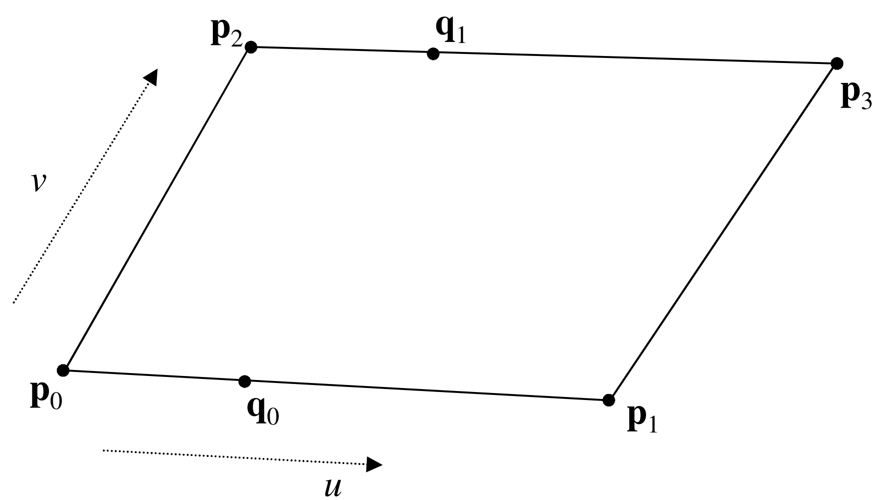
- Control mesh with four points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$
- Compute $\mathbf{x}(u,v)$ using a two-step construction scheme



Bilinear Patch (Step 1)

- For a given value of u, evaluate the linear curves on the two u-direction edges
- Use the same value u for both:

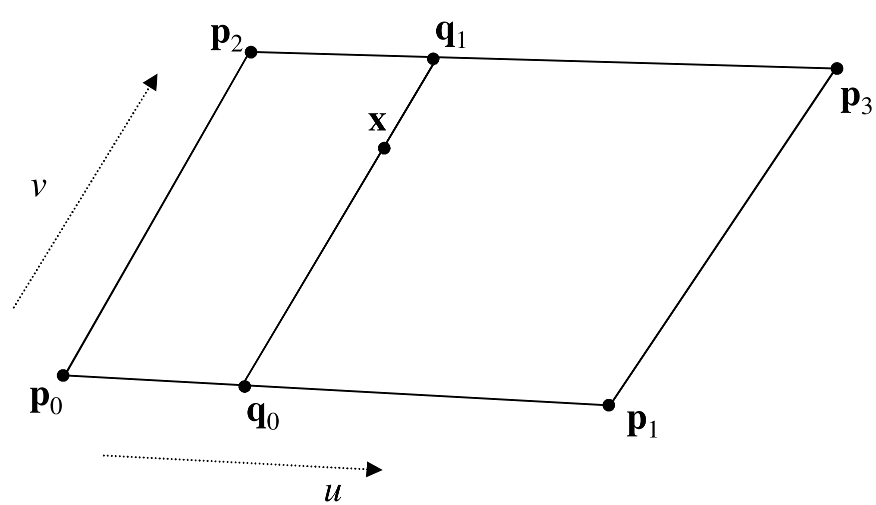
$$\mathbf{q_0} = Lerp(u, \mathbf{p_0}, \mathbf{p_1}) \qquad \mathbf{q_1} = Lerp(u, \mathbf{p_2}, \mathbf{p_3})$$



Bilinear Patch (Step 2)

- Consider that $\mathbf{q_0}$, $\mathbf{q_1}$ define a line segment
- Evaluate it using v to get $\mathbf{x}$

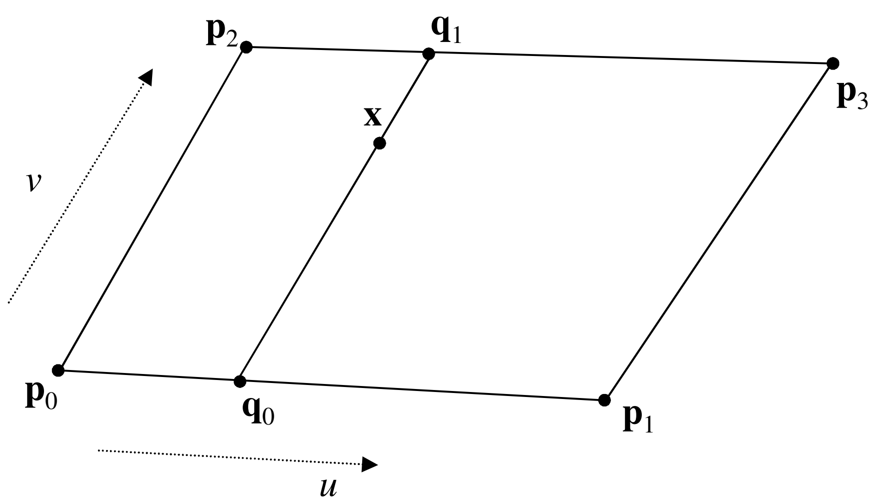
$$\mathbf{x} = Lerp(v, \mathbf{q}_0, \mathbf{q}_1)$$



Bilinear Patch

- Combining the steps, we get the full formula

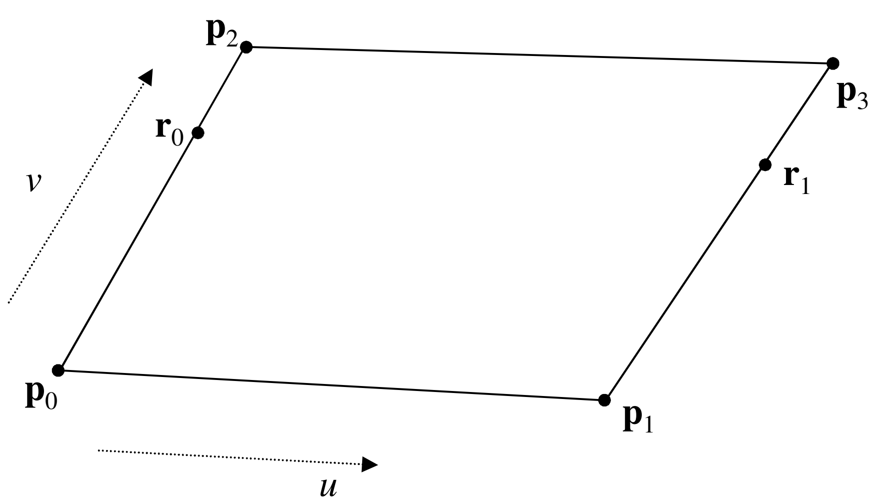
$$\mathbf{x}(u,v) = Lerp(v, Lerp(u,\mathbf{p}_0,\mathbf{p}_1), Lerp(u,\mathbf{p}_2,\mathbf{p}_3))$$



Bilinear Patch

- Try the other order
- Evaluate first in the v direction

$$\mathbf{r}_0 = Lerp(v, \mathbf{p}_0, \mathbf{p}_2) \quad \mathbf{r}_1 = Lerp(v, \mathbf{p}_1, \mathbf{p}_3)$$

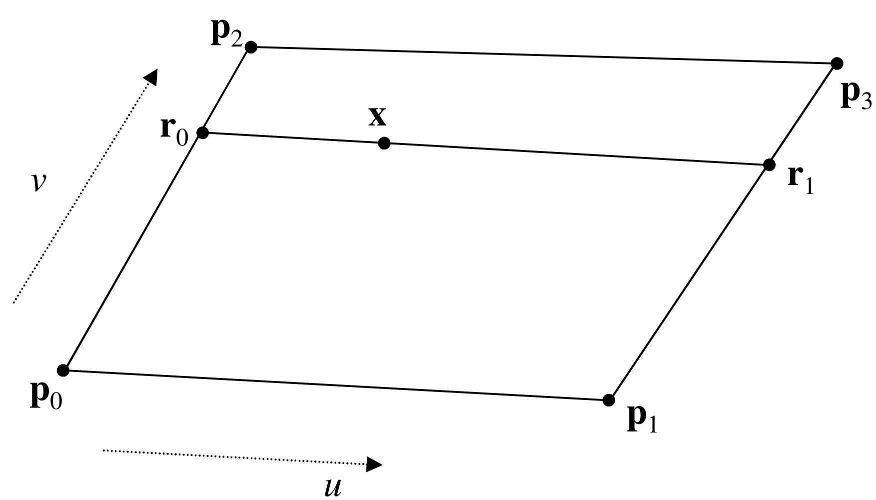


UCSD

Bilinear Patch

- Consider that $\mathbf{r_0}, \mathbf{r_1}$ define a line segment
- Evaluate it using u to get $\mathbf{x}$

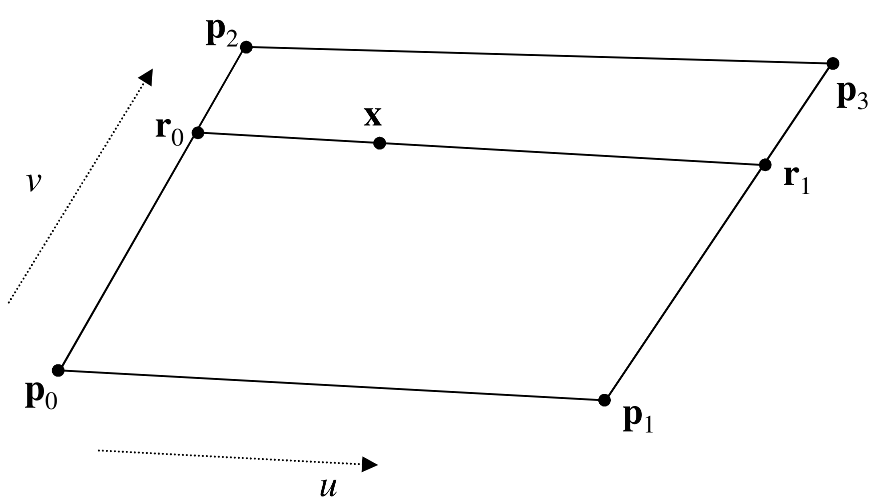
$$\mathbf{x} = Lerp(u, \mathbf{r}_0, \mathbf{r}_1)$$



Bilinear Patch

- The full formula for the v direction first:

$$\mathbf{x}(u,v) = Lerp(u, Lerp(v, \mathbf{p}_0, \mathbf{p}_2), Lerp(v, \mathbf{p}_1, \mathbf{p}_3))$$

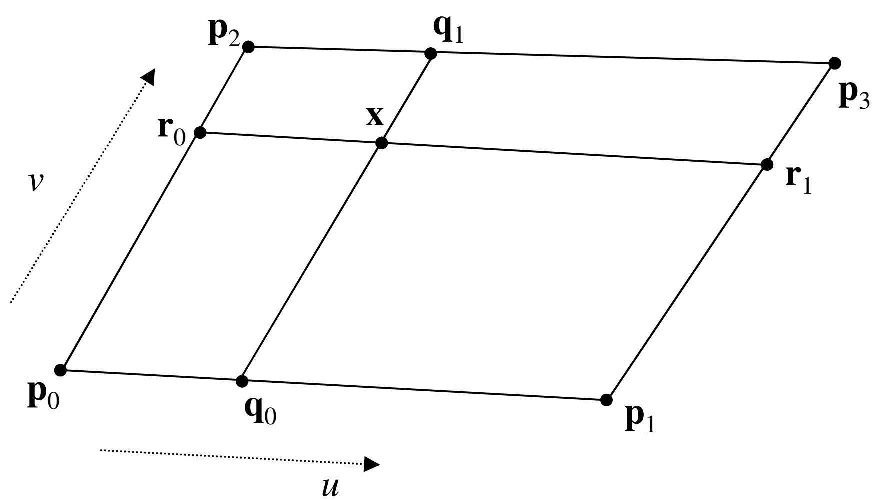


UCSD

Bilinear Patch

- Patch geometry is independent of the order of *u* and *v*

$$\mathbf{x}(u,v) = Lerp(v, Lerp(u, \mathbf{p}_0, \mathbf{p}_1), Lerp(u, \mathbf{p}_2, \mathbf{p}_3))$$
$$\mathbf{x}(u,v) = Lerp(u, Lerp(v, \mathbf{p}_0, \mathbf{p}_2), Lerp(v, \mathbf{p}_1, \mathbf{p}_3))$$



Bilinear Patch

- Visualization