CS559: Computer Graphics

Lecture 21: Subdivision, Bspline, and Texture mapping

Li Zhang

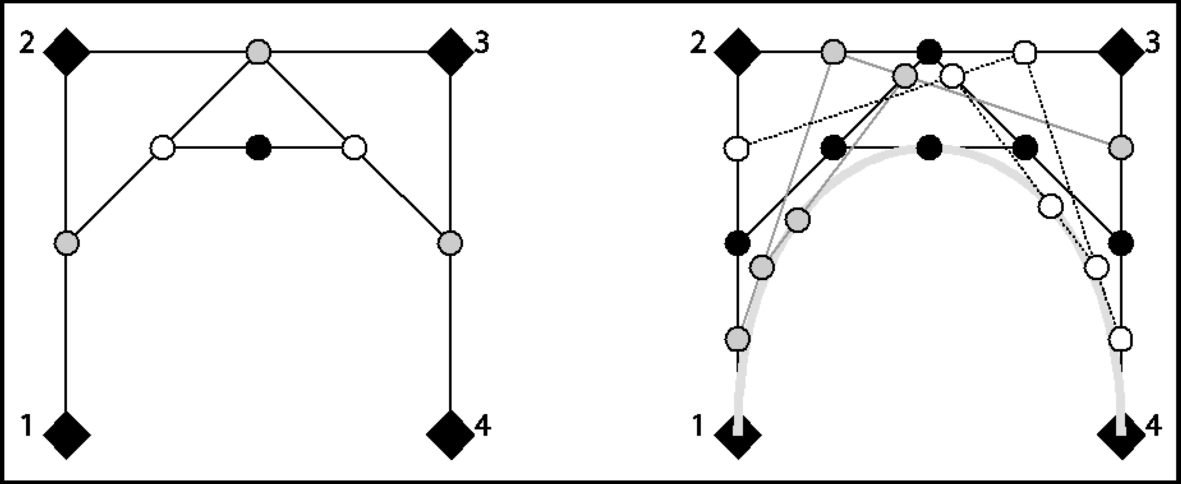
Spring 2008

Today

- Finish on curve modeling
- Start Texture mapping

- Reading
  - Shirley: Ch 15.6.2
  - Redbook: Ch 9
  - (optional) Moller and Haines: *Real-Time Rendering, 3e*, Ch 6
    - Linux: /p/course/cs559-lizhang/public/readings/6_texture.pdf
    - Windows: P:\course\cs559-lizhang\public\readings\6_texture.pdf

Changing u



u=0.5 u=0.25, u=0.5, u=0.75

De Casteljau algorithm, Recursive Rendering

Subdivision

```
DeCasteljau(float p[N][3], float u) {

    for (int n = N-1; n >= 1; --n) {
        for (int j = 0; j < n; ++j) {
            p[j][0] = (1-u) * p[j][0] + u * p[j+1][0];
            p[j][1] = (1-u) * p[j][1] + u * p[j+1][1];
            p[j][2] = (1-u) * p[j][2] + u * p[j+1][2];
        }
    }
    //(p[0][0], p[0][1], p[0][2]) saves the result
}
```

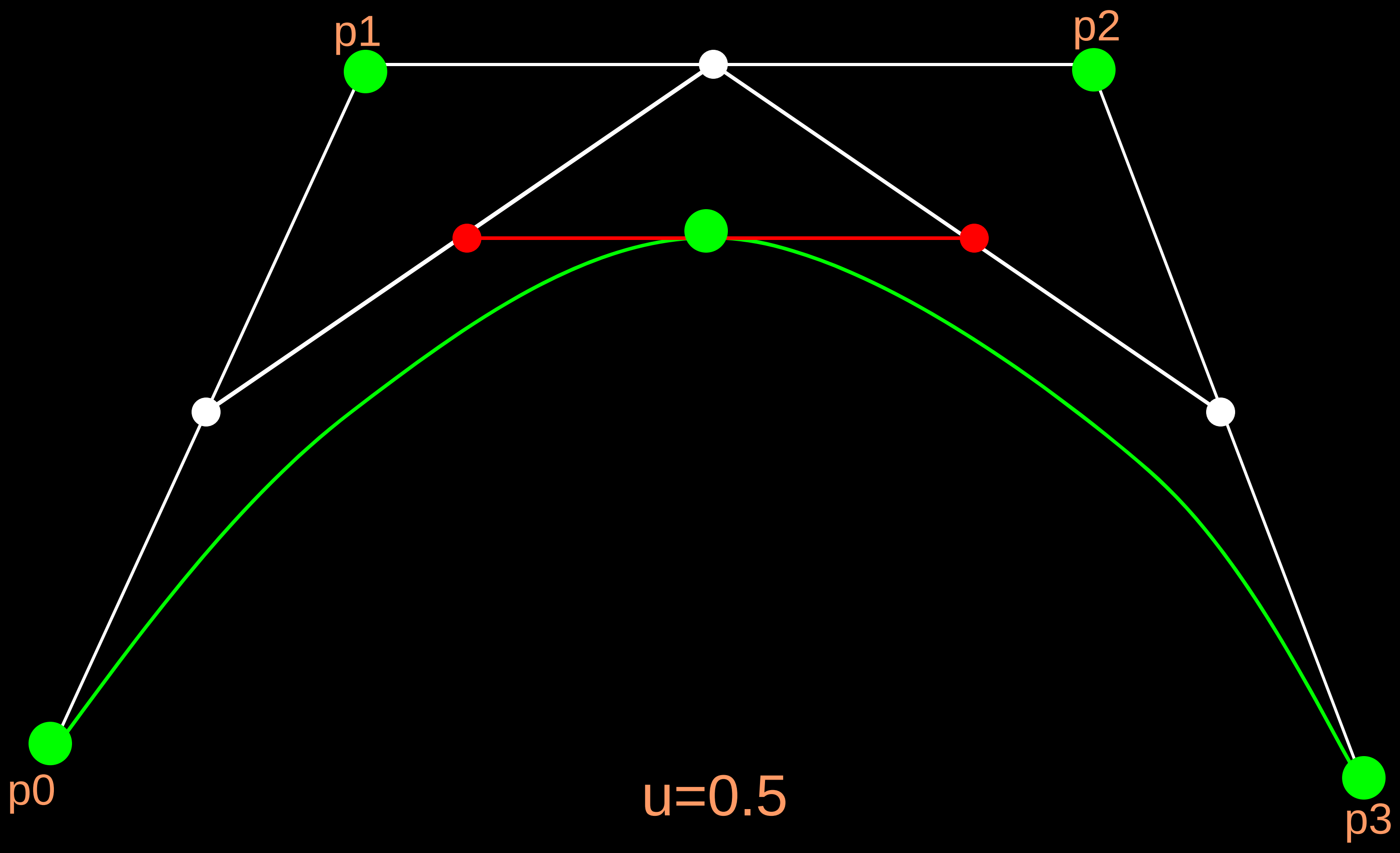
```
DeCasteljau(float p[N][3], float u) {

    for (int n = N-1; n >= 1; --n) {
        for (int j = 0; j < n; ++j) {
            p[j][0] += u*(p[j+1][0]-p[j][0]);
            p[j][1] += u*(p[j+1][1]-p[j][1]);
            p[j][2] += u*(p[j+1][2]-p[j][2]);
        }
    }
    //(p[0][0], p[0][1], p[0][2]) saves the result
}
```

Subdivision

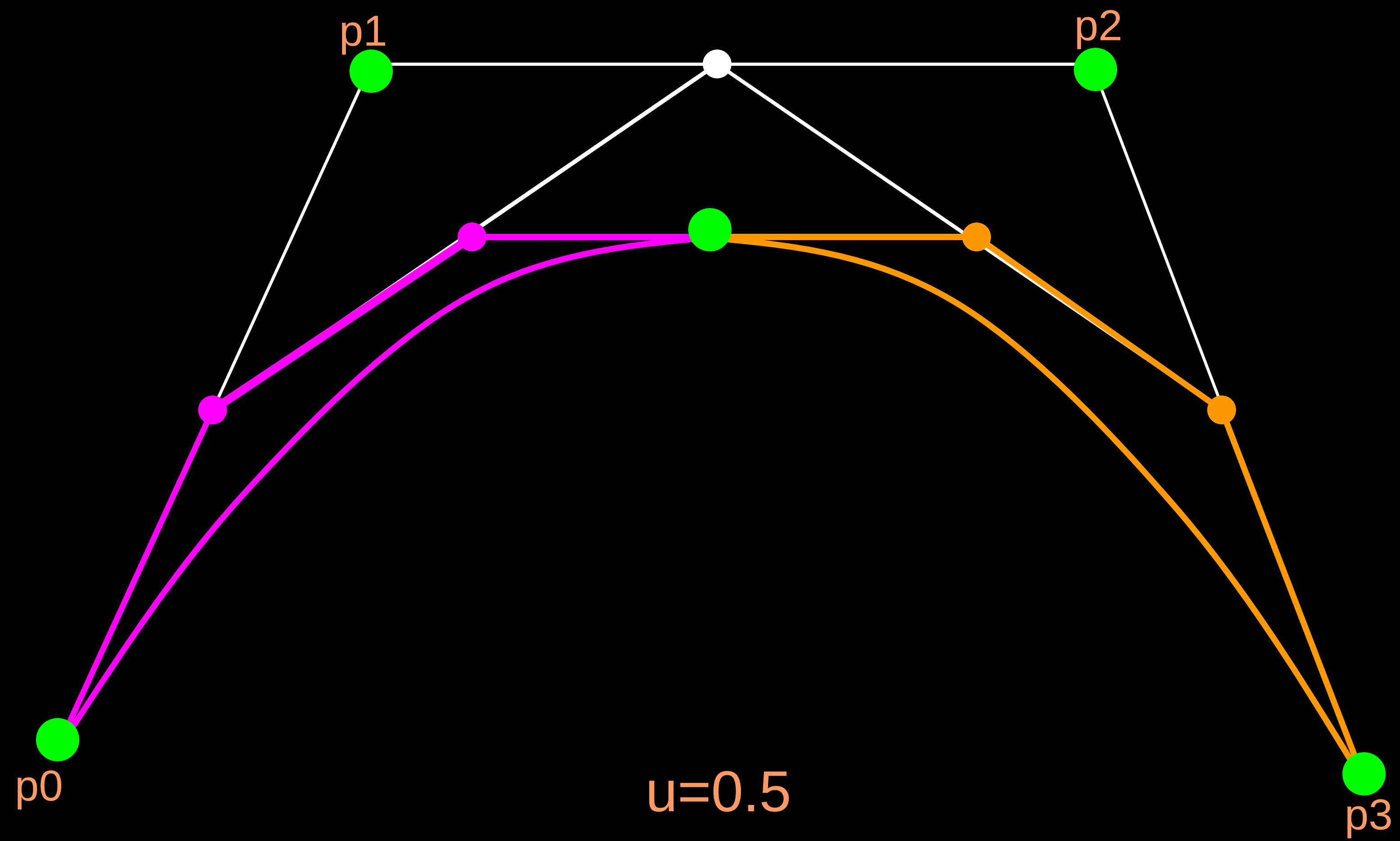
- Given a Bezier curve defined by $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, ..., $\mathbf{P}_n$
- we want to find *two* sets of n+1 control points $\mathbf{Q}_0$, $\mathbf{Q}_1$, $\mathbf{Q}_2$, ..., $\mathbf{Q}_n$ and $\mathbf{R}_0$, $\mathbf{R}_1$, $\mathbf{R}_2$, ..., $\mathbf{R}_n$ such that
  - the Bézier curve defined by $\mathbf{Q}_i$'s is the piece of the original Bézier curve on [0,*u*]
  - the Bézier curve defined by $\mathbf{R}_i$'s is the piece of the original Bézier curve on [*u,1*]

Bezier Curve Subdivision



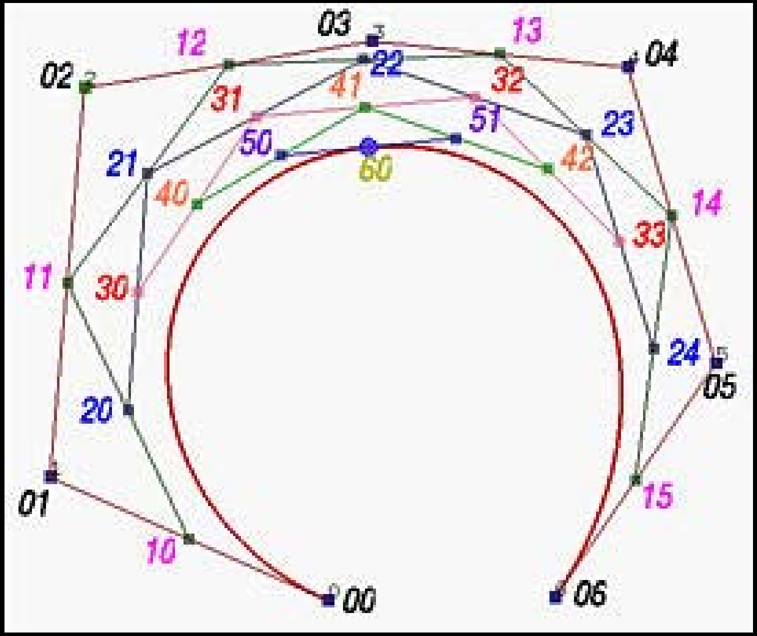
http://www1.cs.columbia.edu/~cs4160/html06f/

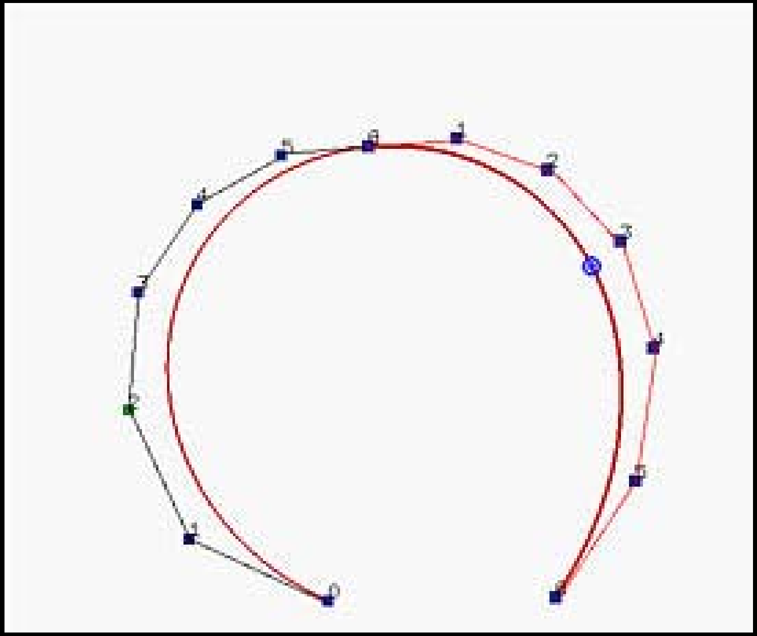
Bezier Curve Subdivision



http://www1.cs.columbia.edu/~cs4160/html06f/

A 6th degree subdivision example



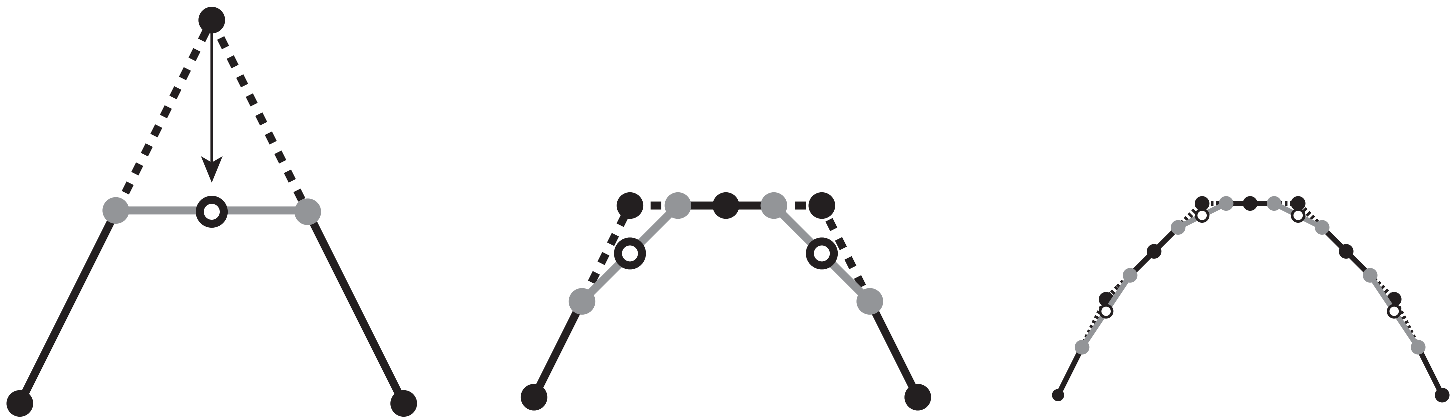


http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/Bezier/bezier-sub.html

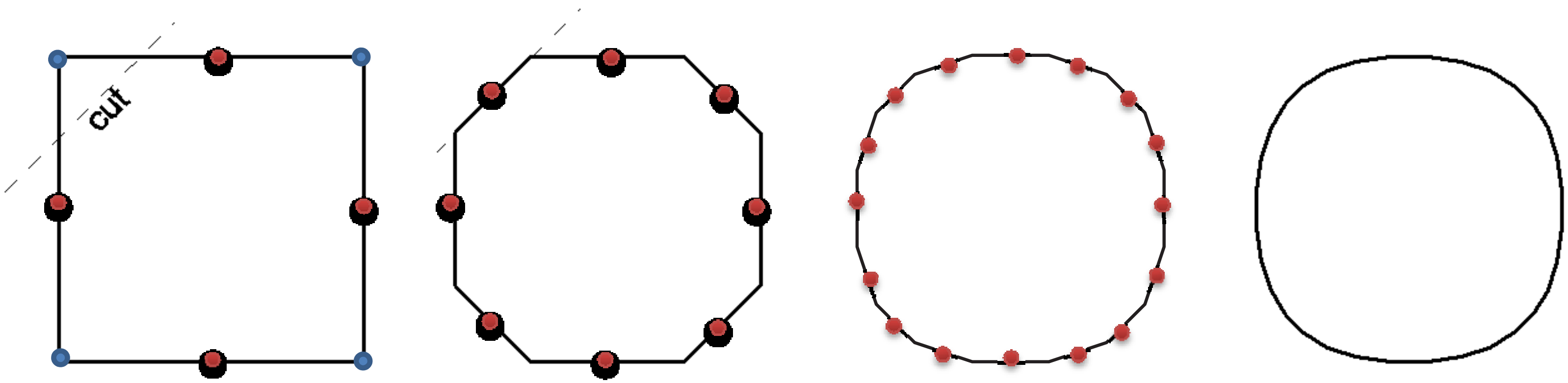
Bezier Curve Subdivision

- Why is subdivision useful?
  - Collision/intersection detection
    - Recursive search
  - Good for curve editing and approximation

Open Curve Approxmiation



Closed Curve Approximation

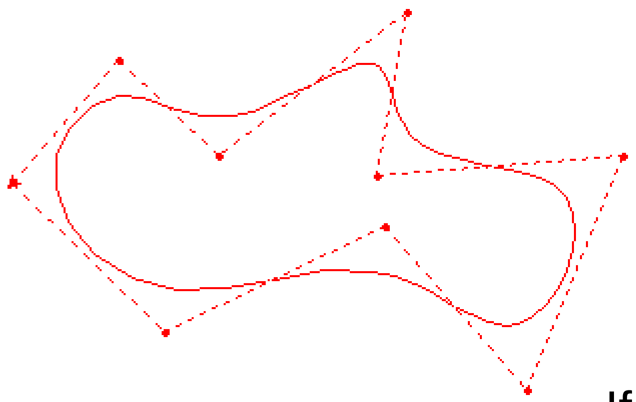


	Interpolate control points	Has local control	C2 continuity
Natural cubics	Yes	No	Yes
Hermite cubics	Yes	Yes	No
Cardinal Cubics	Yes	Yes	No
Bezier Cubics	Yes	Yes	No

	Interpolate control points	Has local control	C2 continuity
Natural cubics	Yes	No	Yes
Hermite cubics	Yes	Yes	No
Cardinal Cubics	Yes	Yes	No
Bezier Cubics	Yes	Yes	No
Bspline Curves	No	Yes	Yes

Bsplines

- Given p1,…pn, define a curve that approximates the curve.

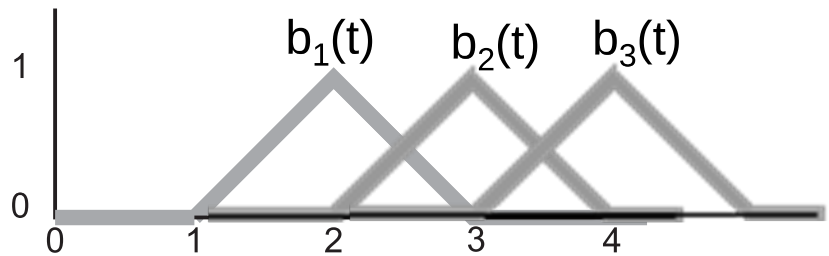


$$\mathbf{f}(t) = \sum_{i=1}^{n} b_i(t)\mathbf{p}_i$$

If $b_i(t)$ is very smooth, so will be **f**

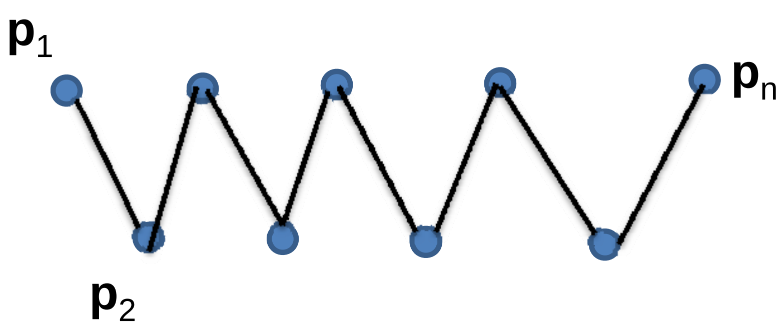
If $b_i(t)$ has local support, **f** will have local control

Uniform Linear B-splines



$$b_{i,2}(t) = b_{0,2}(t-i)$$

$$b_{0,2}(u) = \begin{cases} u & u \in [0,1) \\ 2-u & u \in [1,2) \\ 0 & \text{otherwise} \end{cases}$$



$$\mathbf{f}(t) = \sum_{i=1}^{n} b_i(t)\mathbf{p}_i$$

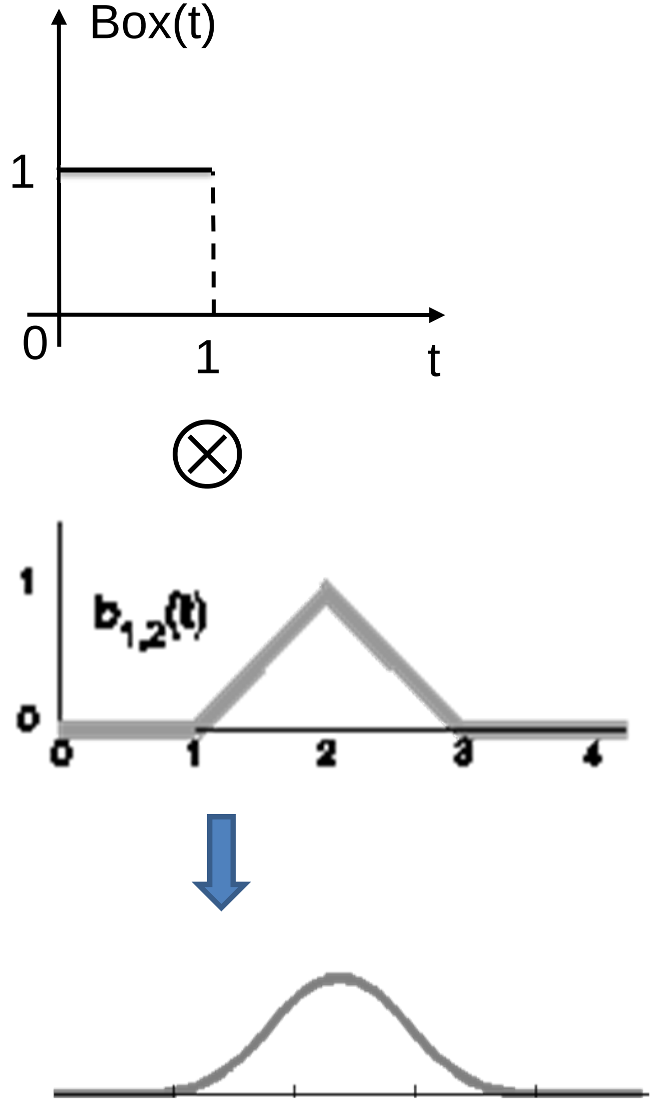
How can we make the curve smooth?

- Convolution/filtering

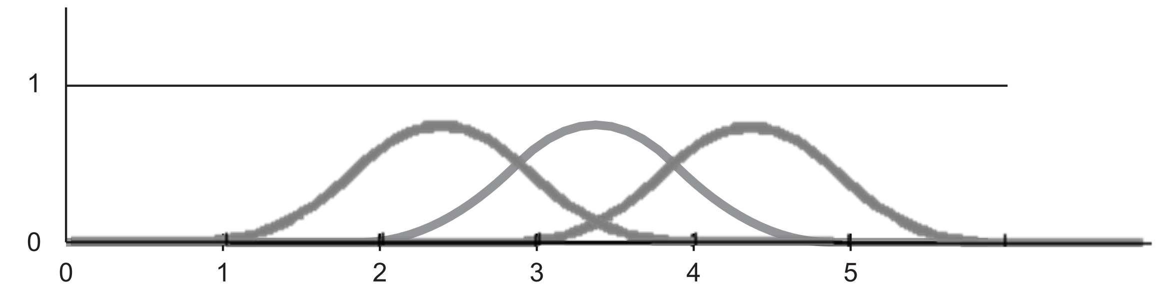
$$\mathbf{f}(t) = \sum_{i=1}^{n} b_i(t)\mathbf{p}_i$$

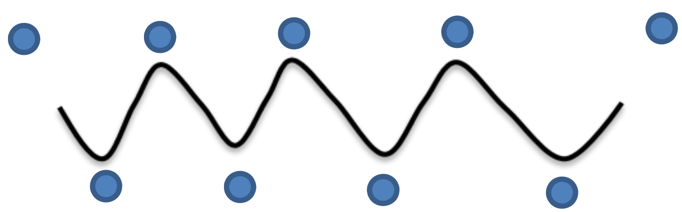
$$\mathbf{f}(t) \otimes \mathrm{Box}(t) = \left(\sum_{i=1}^{n} b_i(t)\mathbf{p}_i \right) \otimes \mathrm{Box}(t)$$

$$= \left(\sum_{i=1}^{n} \left(b_i(t) \otimes \mathrm{Box}(t) \right) \mathbf{p}_i \right)$$



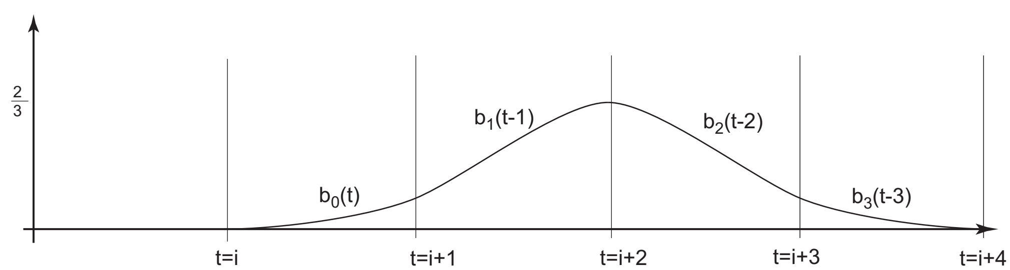
Uniform Quadratic B-splines

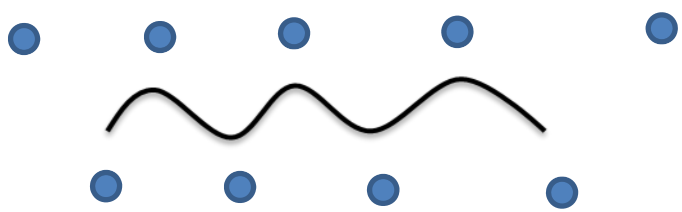




$$\mathbf{f}(t) = \sum_{i=1}^{n} b_i(t)\mathbf{p}_i$$

Uniform Cubic Bspline



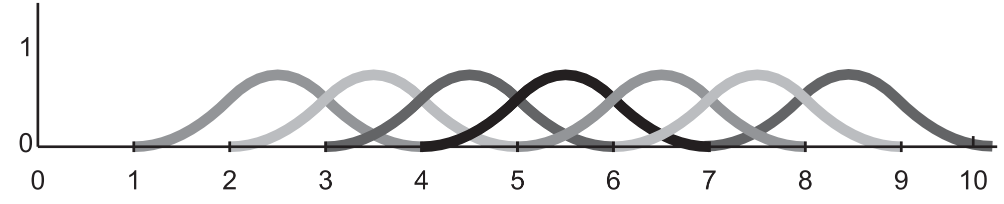


$$\mathbf{f}(t) = \sum_{i=1}^{n} b_i(t)\mathbf{p}_i$$

Uniform B-splines

- Why smoother?
  - Linear = box filter ⊗ box filter
  - Quadric = linear ⊗ box filter
  - Cubic = quadric ⊗ box filter

- Sum = 1 property, translation invariant



- Local control

$$\mathbf{f}(t) = \sum_{i=1}^{n} b_i(t)\mathbf{p}_i$$

- C(k-2) continuity

	Interpolate control points	Has local control	C2 continuity
Natural cubics	Yes	No	Yes
Hermite cubics	Yes	Yes	No
Cardinal Cubics	Yes	Yes	No
Bezier Cubics	Yes	Yes	No
Bspline Curves	No	Yes	Yes

Texture Mapping

Many slides from Ravi Ramamoorthi, Columbia Univ, Greg Humphreys, UVA and Rosalee Wolfe, DePaul tutorial teaching texture mapping visually

Texture Mapping

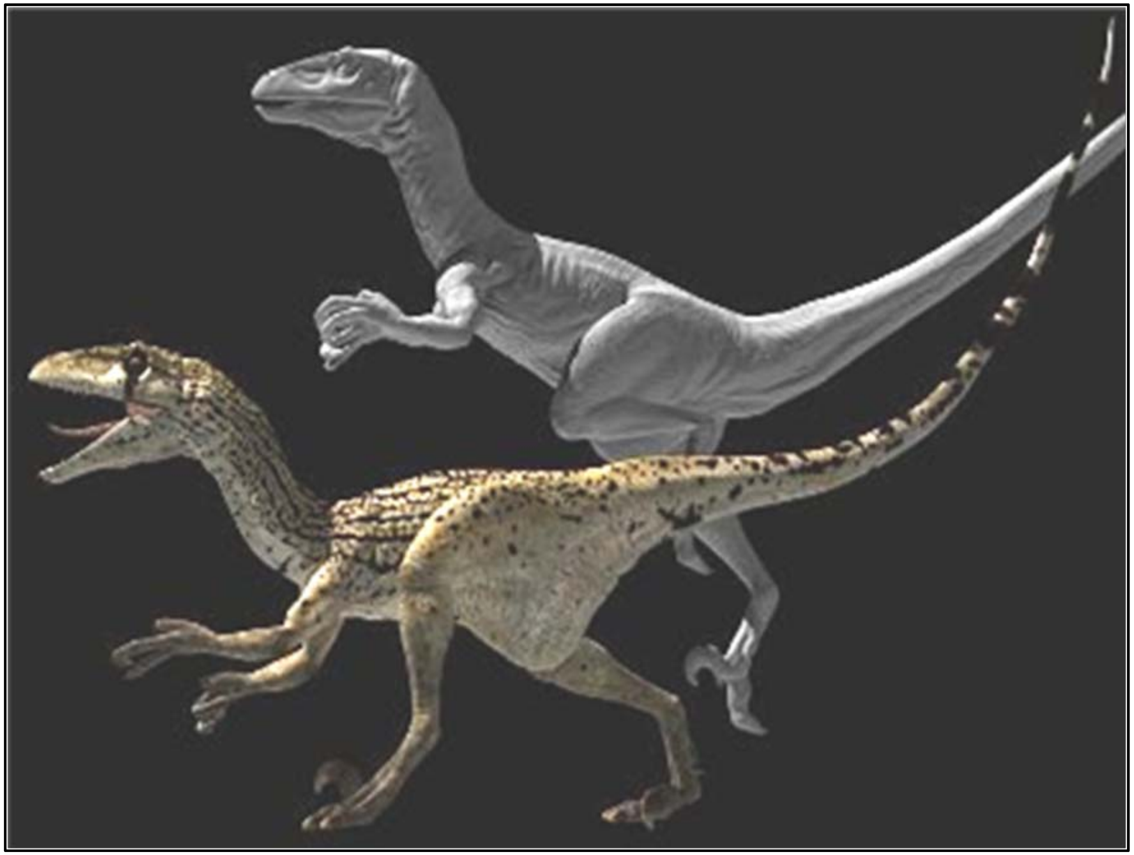
- Important topic: nearly all objects textured
  - Wood grain, faces, bricks and so on
  - Adds visual detail to scenes

Polygonal model

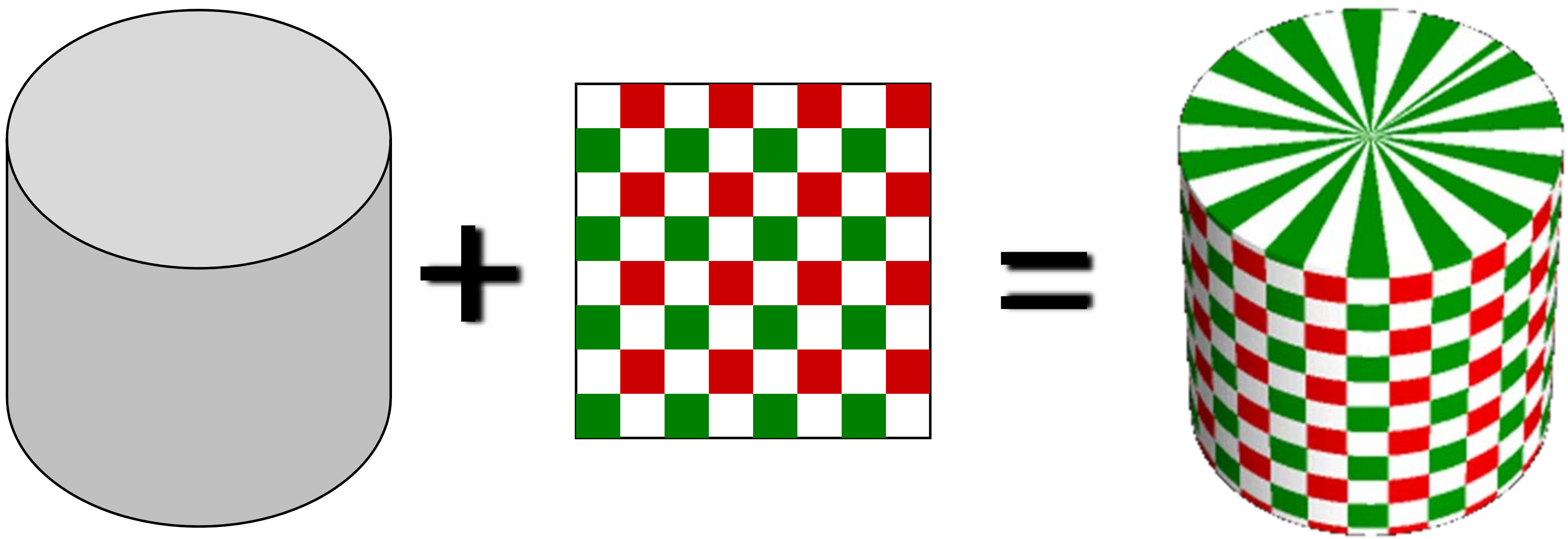
With surface texture

Adding Visual Detail

- Basic idea: use images instead of more polygons to represent fine scale color variation



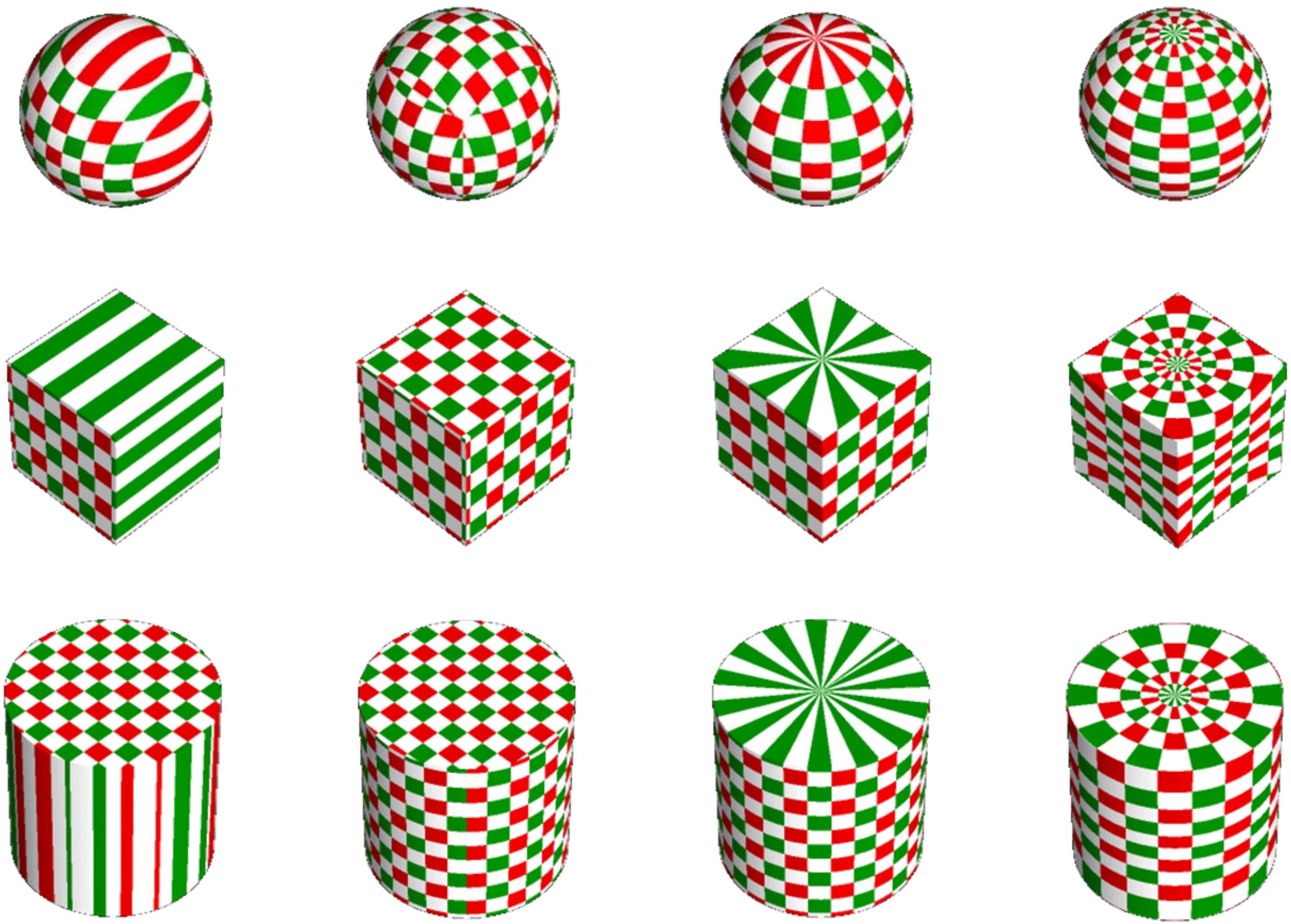
Parameterization



geometry image texture map

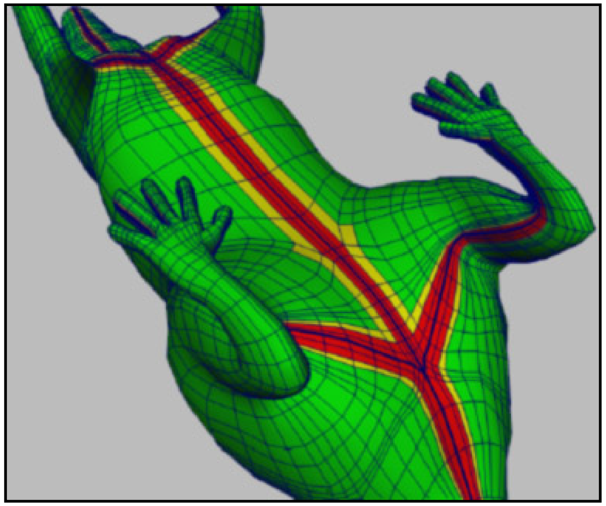
- Q: How do we decide *where* on the geometry each color from the image should go?

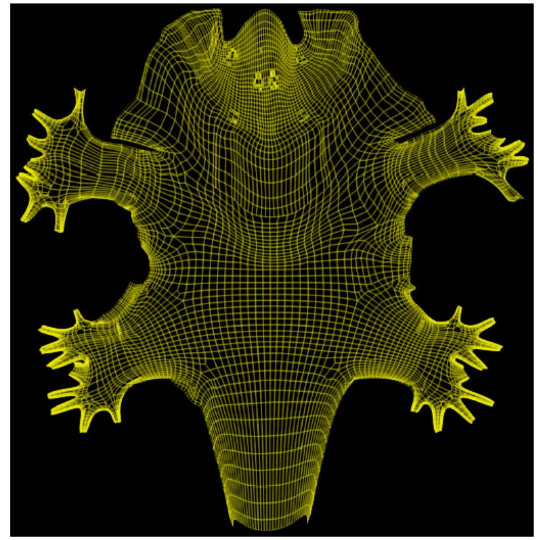
Option: Varieties of mappings



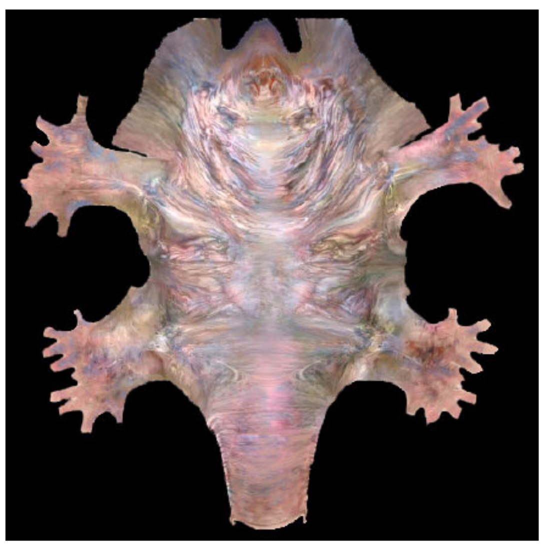
[Paul Bourke]

Option: unfold the surface



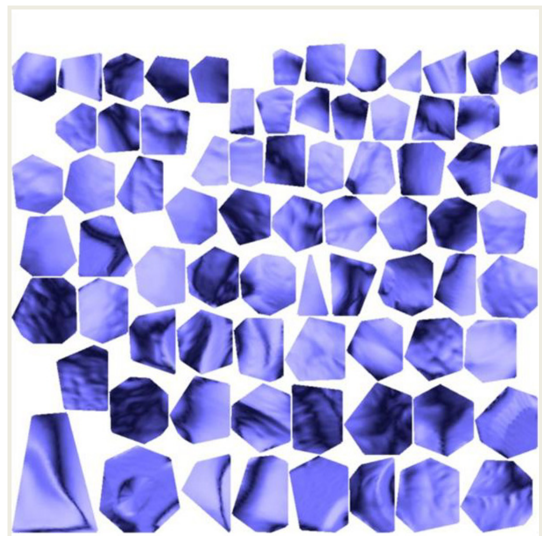


[Piponi2000]

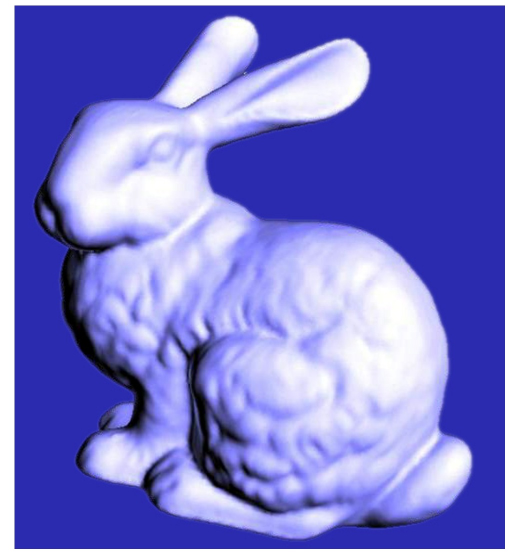


Option: make an atlas

charts



atlas



surface

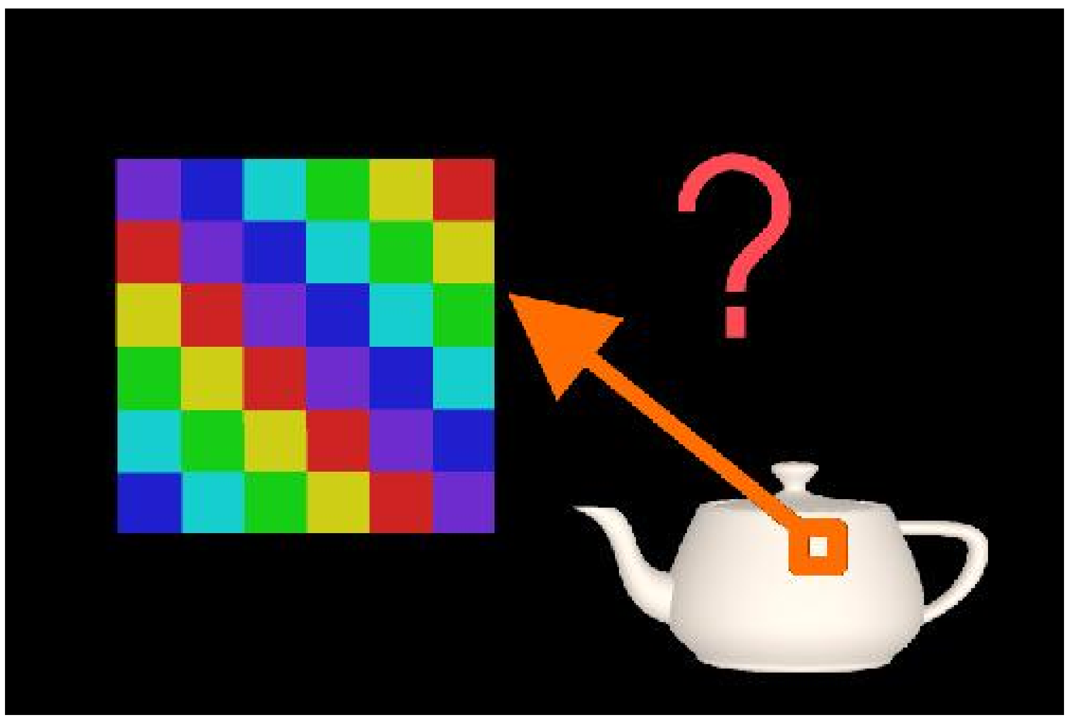
[Sander2001]

Outline

- *Types of mappings*
- Interpolating texture coordinates
- Broader use of textures

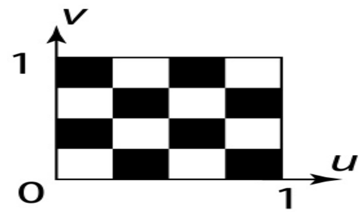
How to map object to texture?

- To each vertex (x,y,z in object coordinates), must associate 2D texture coordinates (s,t)
- So texture fits “nicely” over object

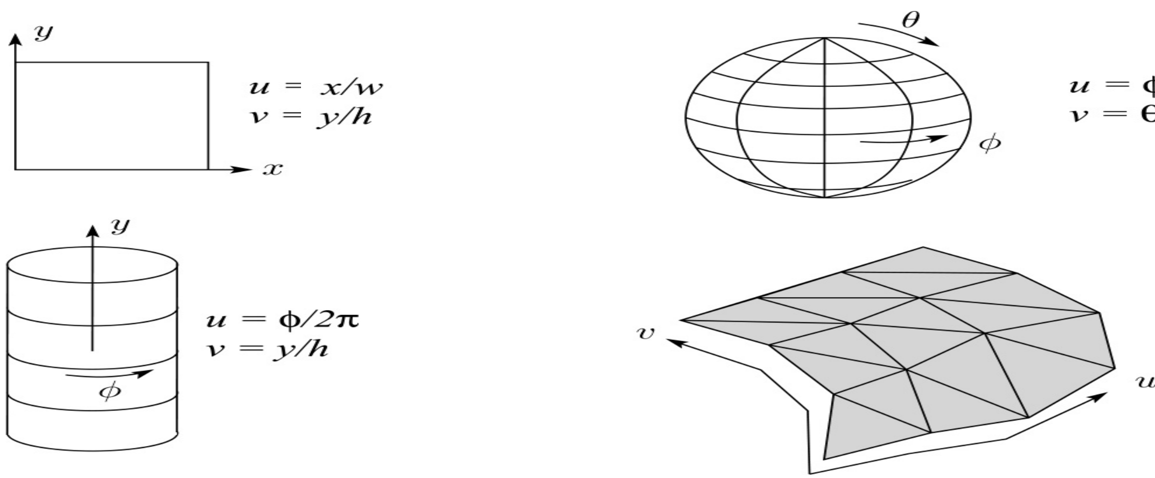


Implementing texture mapping

- A texture lives in it own abstract image coordinates paramaterized by (*u,v*) in the range ([0..1], [0..1]):



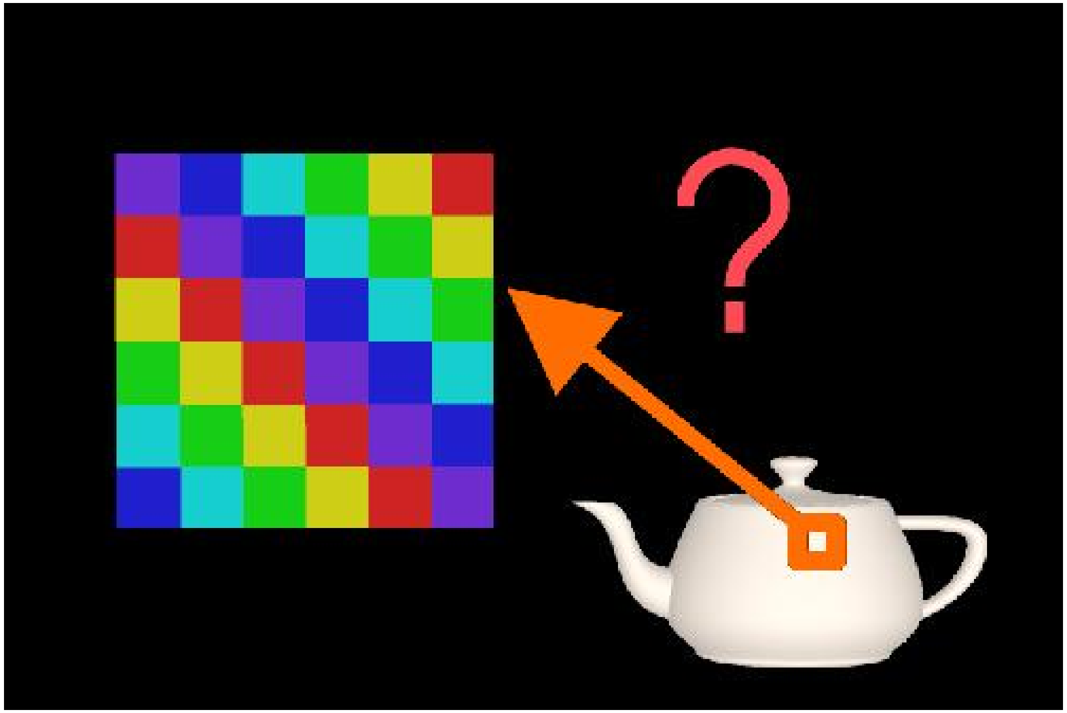
- It can be wrapped around many different surfaces:



- Note: if the surface moves/deforms, the texture goes with it.

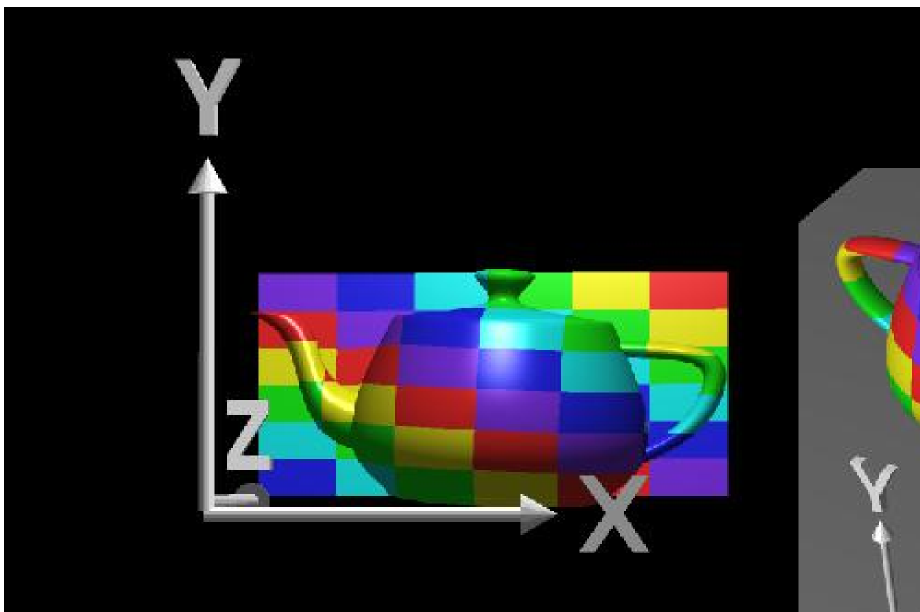
How to map object to texture?

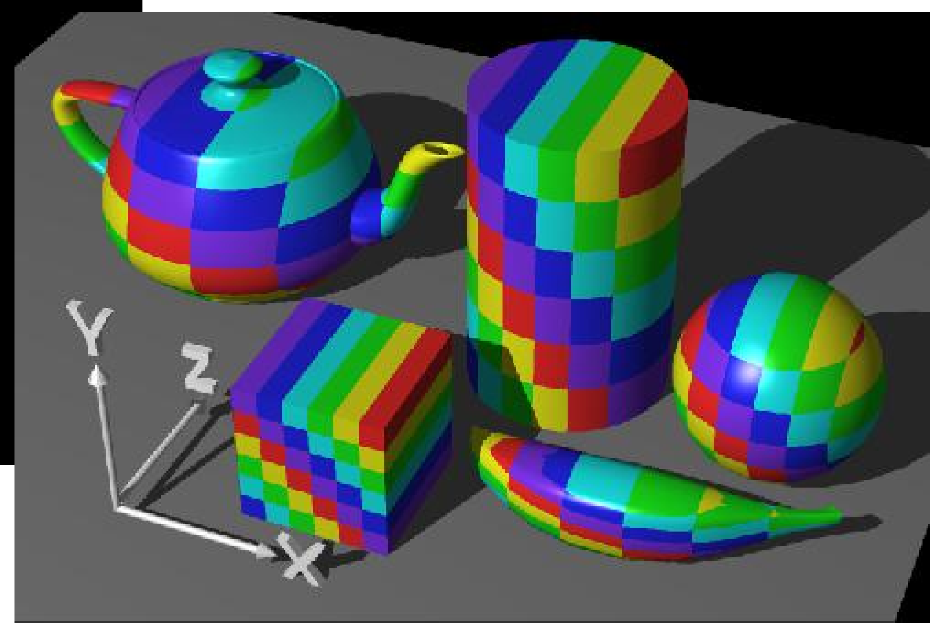
- To each vertex (x,y,z in object coordinates), must associate 2D texture coordinates (s,t)
- So texture fits “nicely” over object



Planar mapping

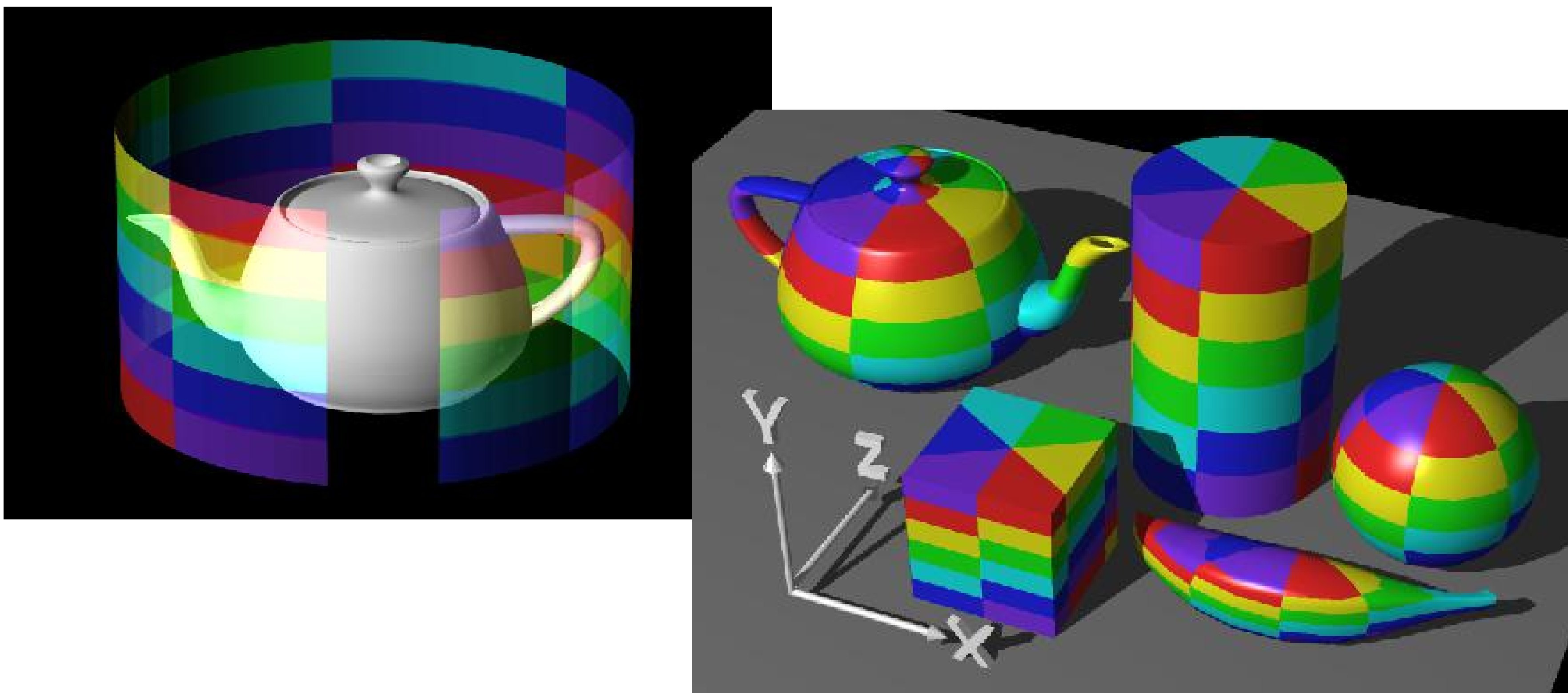
- Like projections, drop z coord (u,v) = (x/W,y/H)
- Problems: what happens near silhouettes?





Cylindrical Mapping

- Cylinder: r, θ, z with (u,v) = (θ/(2π),z)
  - Note seams when wrapping around (θ = 0 or 2π)

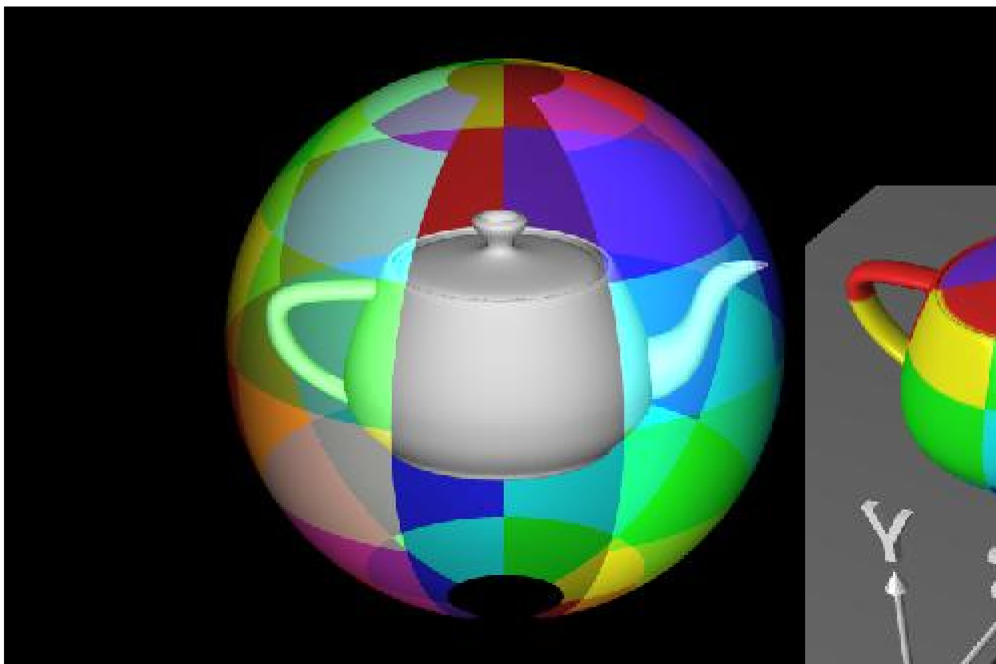


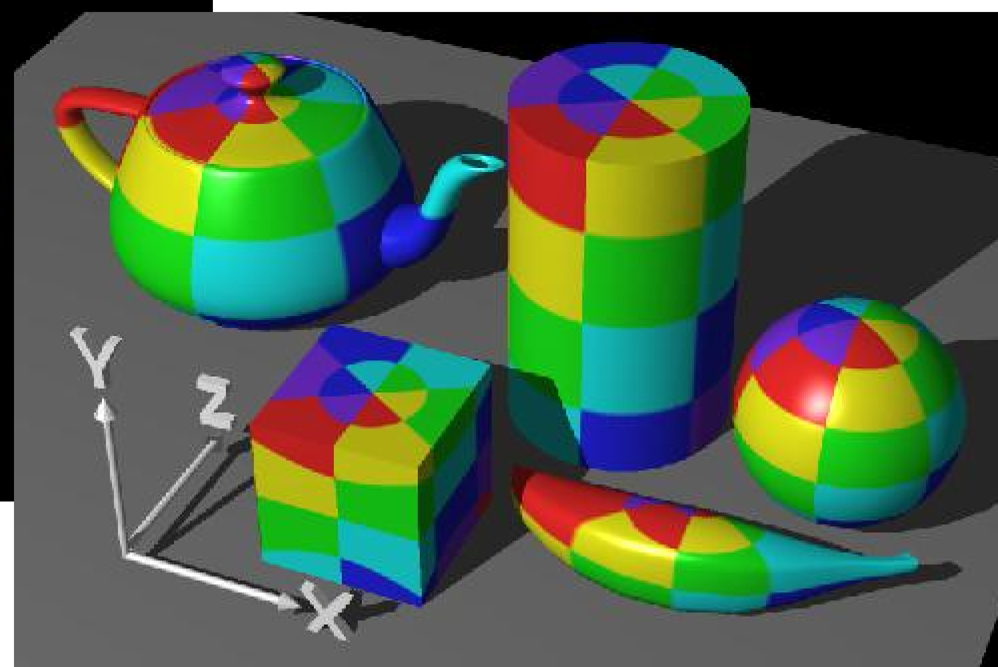
Basic procedure

- First, map (square) texture to basic map shape
- Then, map basic map shape to object
  - Or vice versa: Object to map shape, map shape to square
- Usually, this is straightforward
  - Maps from square to cylinder, plane, …
  - Maps from object to these are simply coordinate transform

Spherical Mapping

- Convert to spherical coordinates: use latitude/long.
  - Singularities at north and south poles





Cube Mapping