Computer Graphics

-Curves and Surfaces-

Oliver Bimber

Course Schedule

Type	Date	Time	Room	Topic	Comment
C1	01.03.2016	13:45-15:15	HS 18	Introduction and Course Overview	Conference
C2	15.03.2016	13:45-15:15	HS 18	Transformations and Projections	Easter Break
C3	05.04.2016	13:45-15:15	HS 18	Raster Algorithms and Depth Handling	
C4	12.04.2016	13:45-15:15	HS 18	Local Shading and Illumination	
C5	19.04.2016	13:45-15:15	HS 18	Texture Mapping Basics	
C6	26.4.2016	13:45-15:15	HS 18	Advanced Texture Mapping & Graphics Pipelines	
C7	03.05.2016	13:45-15:15	HS 18	Intermediate Exam	
C8	09.05.2016	17:15-18:45	HS 18	Global Illumination I: Raytracing	
C9	10.05.2016	13:45-15:15	HS 18	Global Illumination II: Radiosity	Conference / Holiday
C10	31.05.2016	13:45-15:15	HS 18	Volume Rendering	
C11	07.06.2016	13:45-15:15	HS 18	Scientific Data Visualization	
C12	14.06.2016	13:45-15:15	HS 18	Curves and Surfaces	
C13	21.06.2016	13:45-15:15	HS 18	Basics of Animation	
C14	28.06.2016	13:45-15:15	HS 18	Final Exam	
C15	04.10.2016	13:45-15:15	TBA	Retry Exam	

NEXT ICG LAB TALK: JUNE 14, 2016, 4:30PM

Institute of Computer Graphics

Prof. Thomas Pock

Graz University of Technology

Learning better models for computer vision

Science Park 3

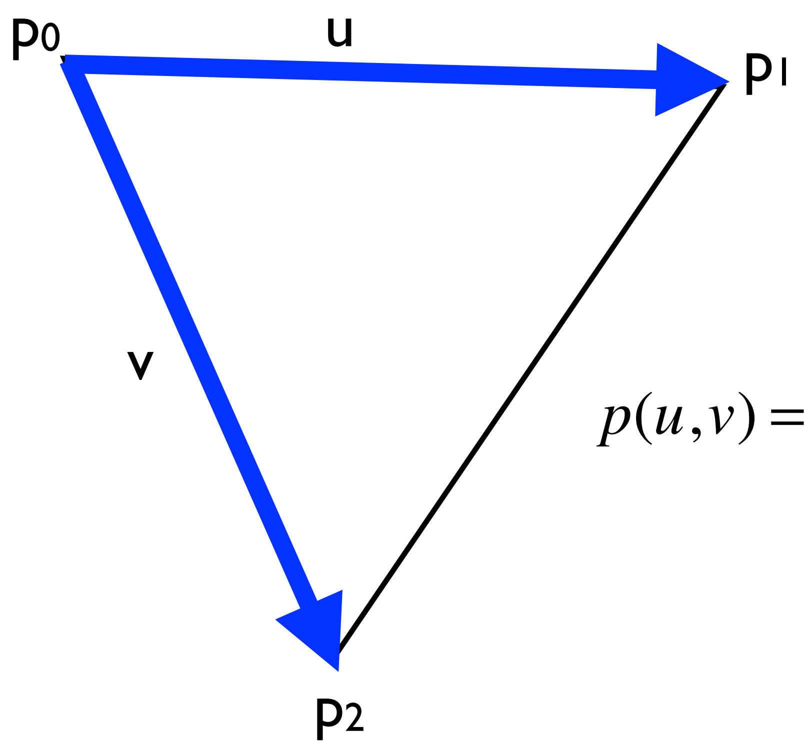
Room S3 048

JKU

For more information about our talks visit
http://www.cg.jku.at/talks/invited

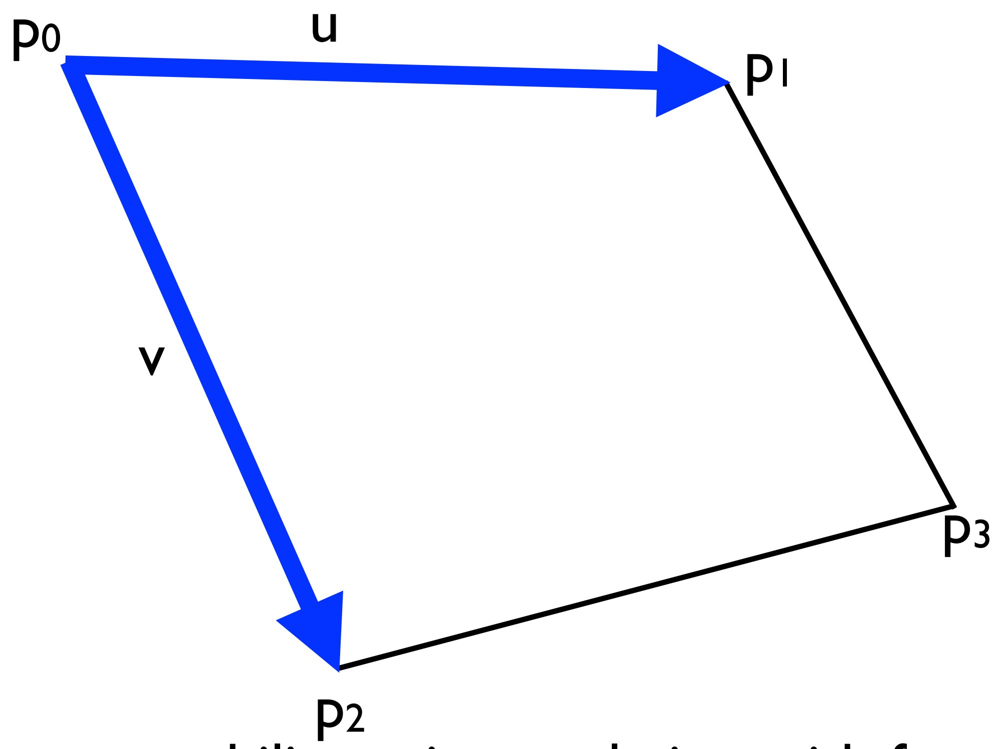
Recap: Bilinear Interpolation

- Bilinear interpolation in triangles was essential for many components of shading and rasterization
- Bilinear interpolation can be extended to more vertices (let‘s call them control points), such as patches (quads)
- The four vertices can be non-coplanar (i.e., their 3D coordinates are not on the same plane)
- Interpolating 3D points over parameters u,v allows computing intermediate 3D points on patches (smoothly)



$$p(u,v) = (1-u-v)p_0 + up_1 + vp_2$$

bilinear interpolation in triangle

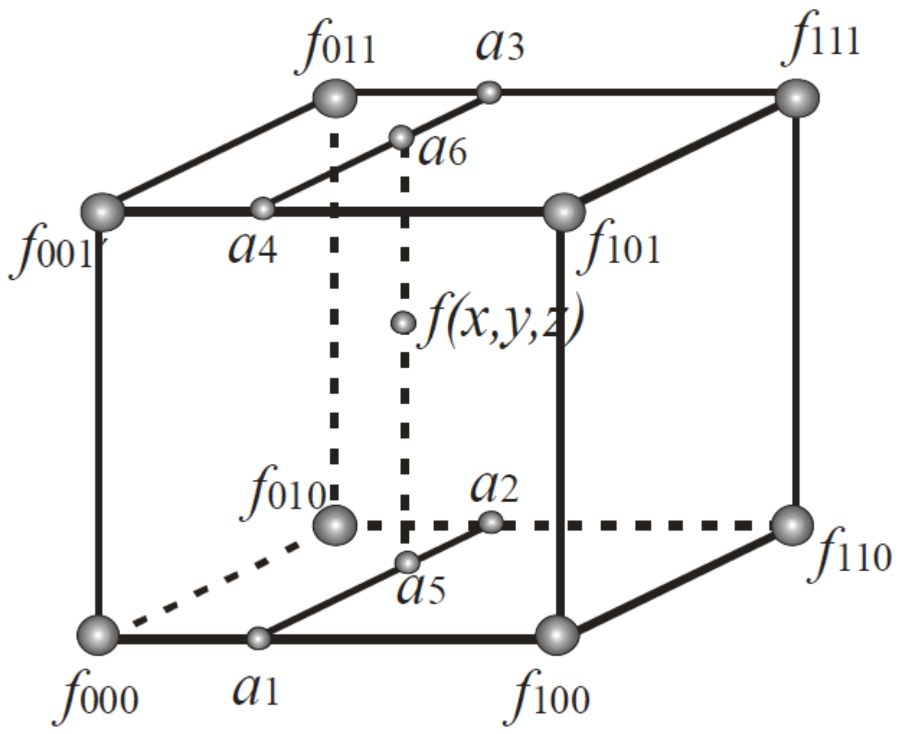


$$p(u,v) = \\ (1-u)(1-v)p_0 + \\ u(1-v)p_1 + \\ v(1-u)p_2 + \\ uvp_3$$

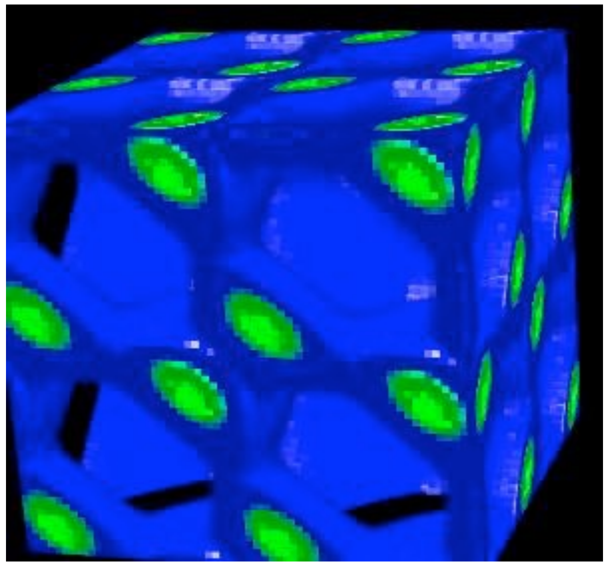
bilinear interpolation with four non-coplanar vertices

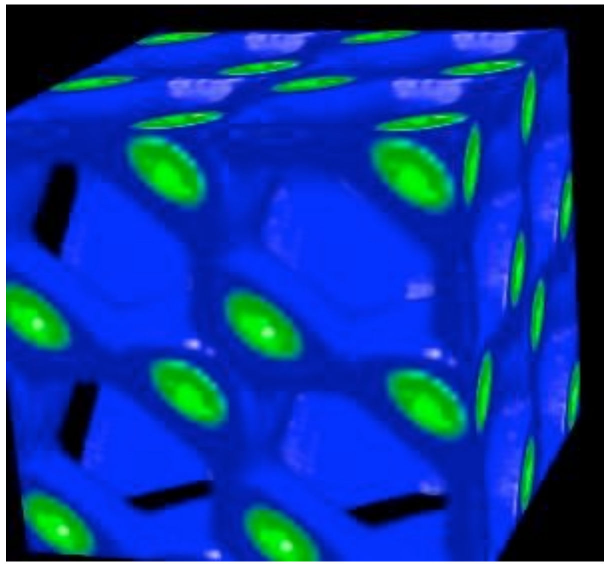
Recap: Trilinear Interpolation

- We have discussed interpolation in triangles already
- Instead of using the voxel values directly (which would equal a nearest neighbour selection for intermediate volume positions), trilinear interpolation leads to smoother results and less aliasing
- Intermediate values are interpolated from the eight voxel vales at the corners of each cell
- Note, that each cell is assumed to be a unit-cell (i.e., x,y,z are normalized to 0..1)
- The voxel values at each corner are indicated with f in this example



$$a_1 = (1-x)f_{000} + xf_{100}, a_2 = (1-x)f_{010} + xf_{110}$$
$$a_3 = (1-x)f_{011} + xf_{111}, a_4 = (1-x)f_{001} + xf_{101}$$
$$a_5 = (1-y)a_1 + ya_2, a_6 = (1-y)a_4 + ya_3$$
$$f(x,y,z) = (1-z) * a_5 + za_6$$

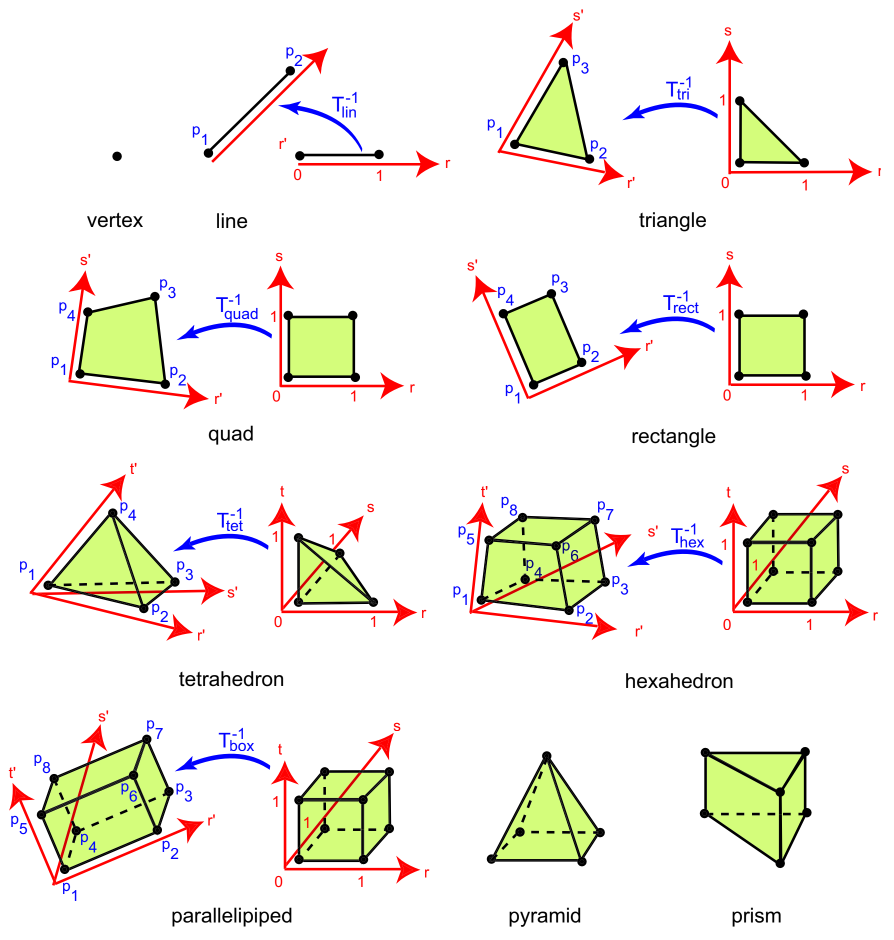

nearest neighbour


trilinear interpolation

Recap: Various Cell Types

- Constant basis functions are simple and have virtually no computational cost, but provide only low quality approximations
- Higher-order basis functions provide more continuos reconstruction of the original data
- Linear basis functions are the next best choice, but they require some knowledge about the cell types used in the grid
  - example: trilinear interpolation in cube-like cell (defined by its 8 vertices)
- But cube-like cells are not the only cell choice - different types exist
- Vertices are sample points

$$\tilde{f} = \sum_{i=1}^{N} f_i \phi_i$$



cell types in world and reference coordinate systems

$$\phi_1^0(r) = 1$$

vertex

$$\phi_1^1(r) = 1 - r \qquad \phi_2^1(r) = r$$

line

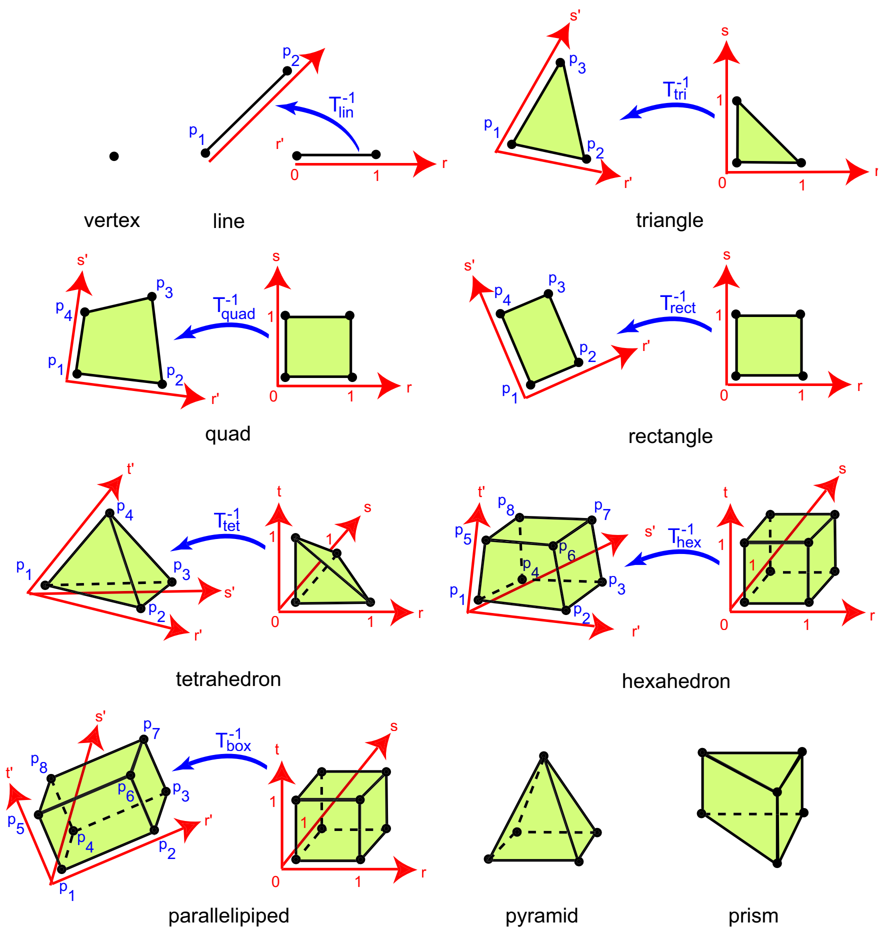
$$\phi_1^1(r,s) = 1 - r - s \quad \phi_2^1(r,s) = r \quad \phi_3^1(r,s) = s$$

triangle

Recap: Various Cell Types

- Constant basis functions are simple and have virtually no computational cost, but provide only low quality approximations
- Higher-order basis functions provide more continuos reconstruction of the original data
- Linear basis functions are the next best choice, but they require some knowledge about the cell types used in the grid
    - example: trilinear interpolation in cube-like cell (defined by its 8 vertices)
- But cube-like cells are not the only cell choice - different types exist
- Vertices are sample points

$$\tilde{f} = \sum_{i=1}^{N} f_i \phi_i$$



cell types in world and reference coordinate systems

$$\phi_1^1(r,s) = (1-r)(1-s) \qquad \phi_2^1(r,s) = r(1-s)$$
$$\phi_3^1(r,s) = rs \qquad \phi_4^1(r,s) = (1-r)s$$

quad

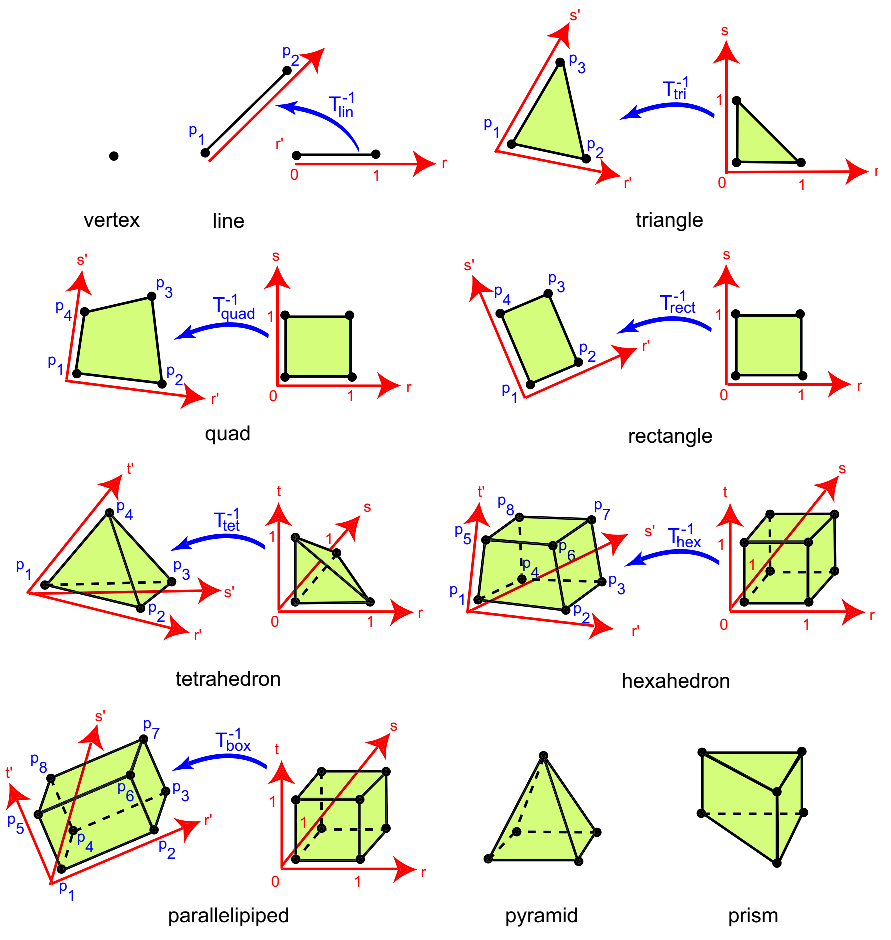
$$\phi_1^1(r,s,t) = 1-r-s-t \qquad \phi_2^1(r,s,t) = r$$
$$\phi_3^1(r,s,t) = s \qquad \phi_4^1(r,s,t) = t$$

tetrahedron

Recap: Various Cell Types

- Constant basis functions are simple and have virtually no computational cost, but provide only low quality approximations
- Higher-order basis functions provide more continuos reconstruction of the original data
- Linear basis functions are the next best choice, but they require some knowledge about the cell types used in the grid
  - example: trilinear interpolation in cube-like cell (defined by its 8 vertices)
- But cube-like cells are not the only cell choice - different types exist
- Vertices are sample points

$$\tilde{f} = \sum_{i=1}^{N} f_i \phi_i$$



cell types in world and reference coordinate systems

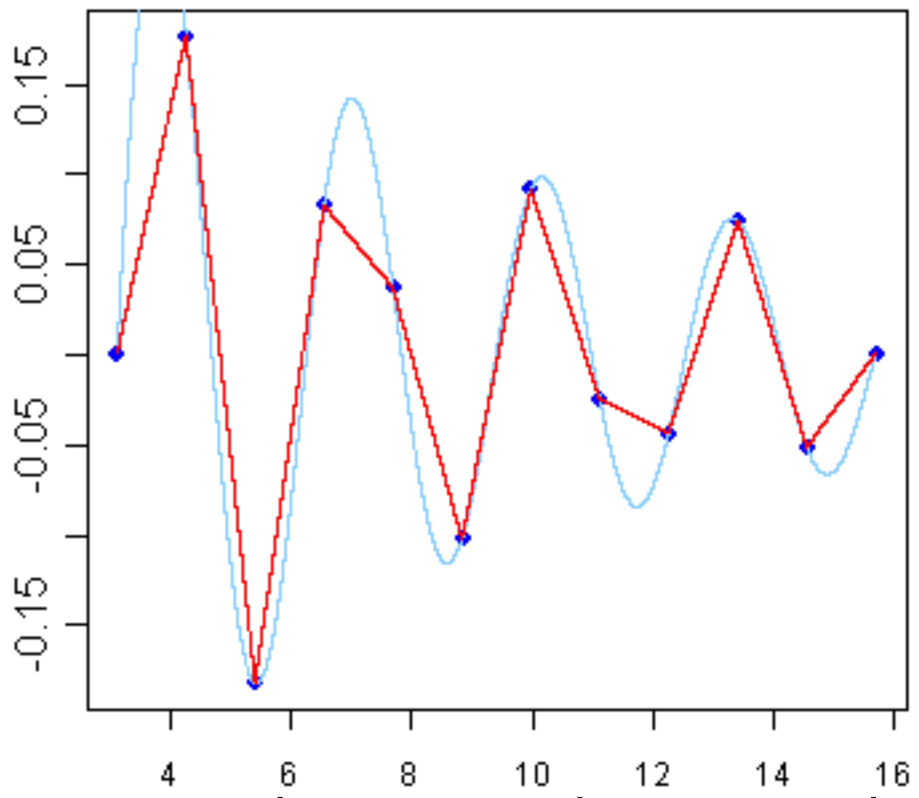
$$\phi_1^1(r,s,t) = (1-r)(1-s)(1-t)$$
$$\phi_2^1(r,s,t) = r(1-s)(1-t)$$
$$\phi_3^1(r,s,t) = rs(1-t)$$
$$\phi_4^1(r,s,t) = (1-r)s(1-t)$$

hexahedron

$$\phi_5^1(r,s,t) = (1-r)(1-s)t$$
$$\phi_6^1(r,s,t) = r(1-s)t$$
$$\phi_7^1(r,s,t) = rst \quad \phi_8^1(r,s,t) = (1-r)st$$

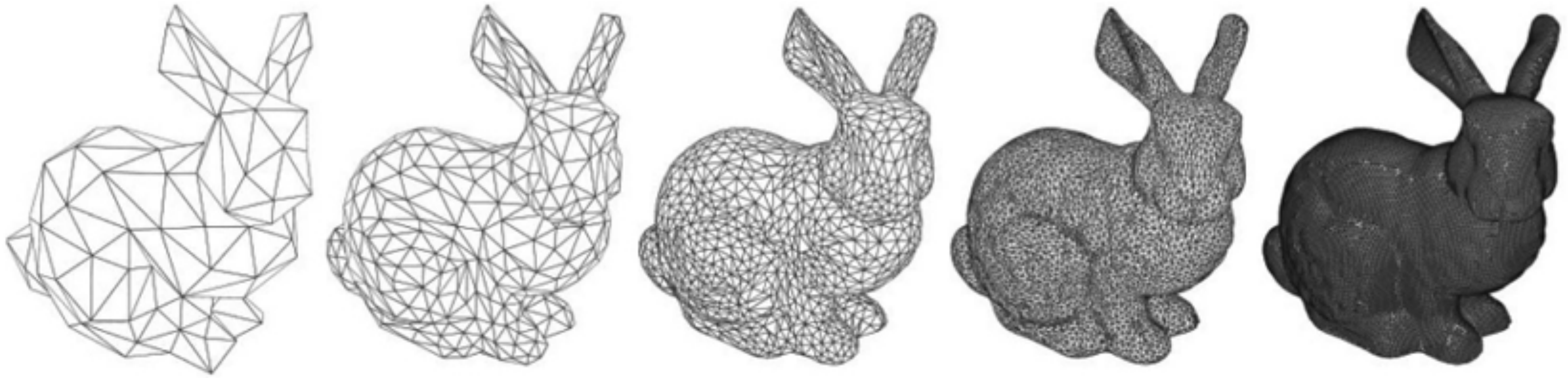
Piece-Wise Linear Interpolation

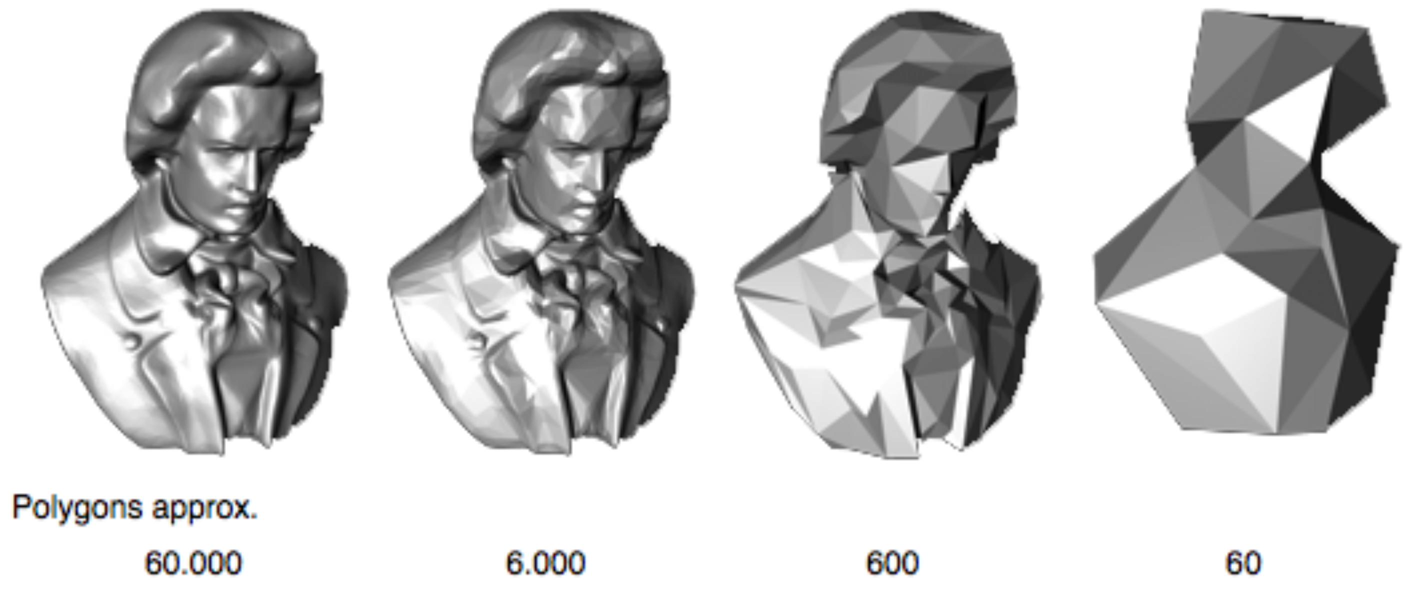
- Thus far, we used linear functions for interpolation!
- More complex geometry was stitched from multiple pieces (e.g. cells, such as cubes, triangles, line segments, etc.) that are all interpolated independently (piece-wise linear interpolation)



piece-wise linear vs. non-linear interpolation of a function

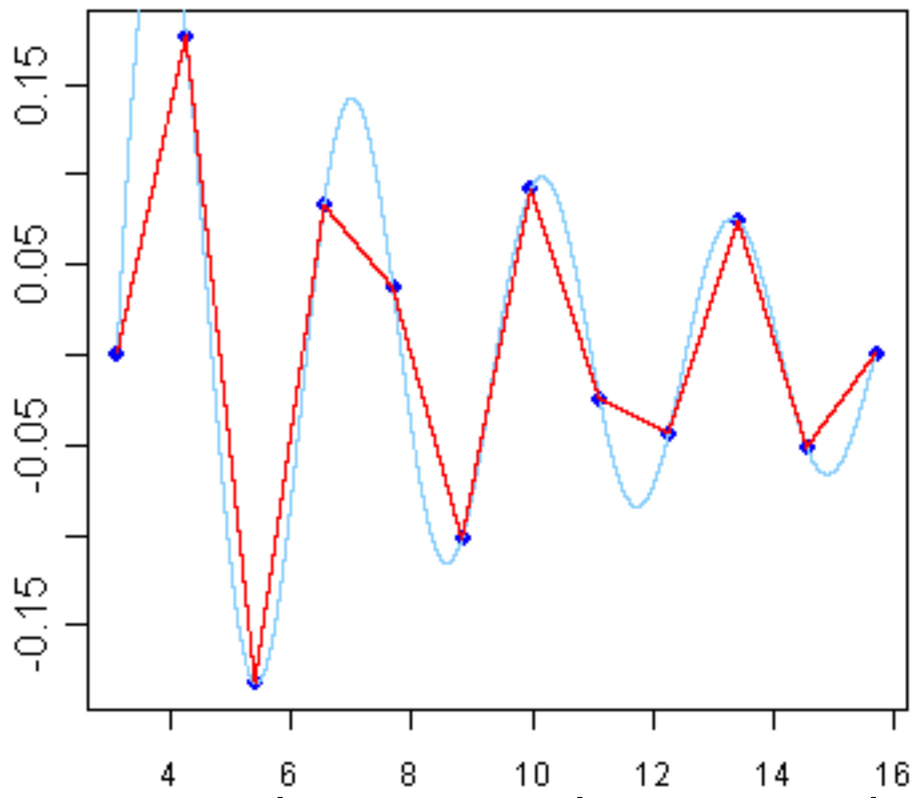
Triangle Meshes





Piece-Wise Linear Interpolation

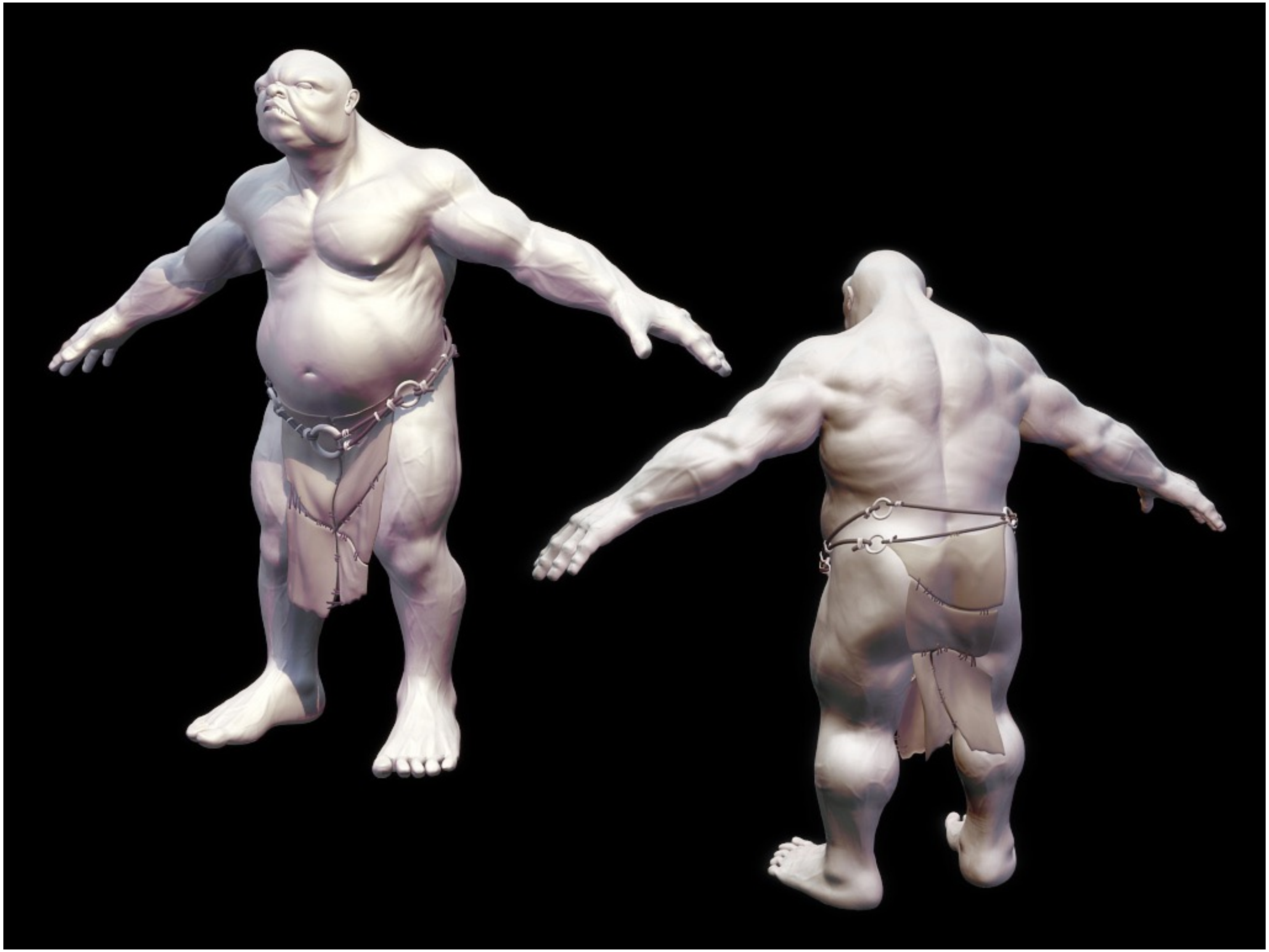
- Thus far, we used linear functions for interpolation!
- More complex geometry was stitched from multiple pieces (e.g. cells, such as cubes, triangles, line segments, etc.) that are all interpolated independently (piece-wise linear interpolation)
- Piece-wise linear interpolation has limited potential for describing complex surfaces, since they would have to be composed from many small patches/cells to avoid visible discontinuities at the edges



piece-wise linear vs. non-linear interpolation of a function

Complex Surfaces

Complex Surfaces

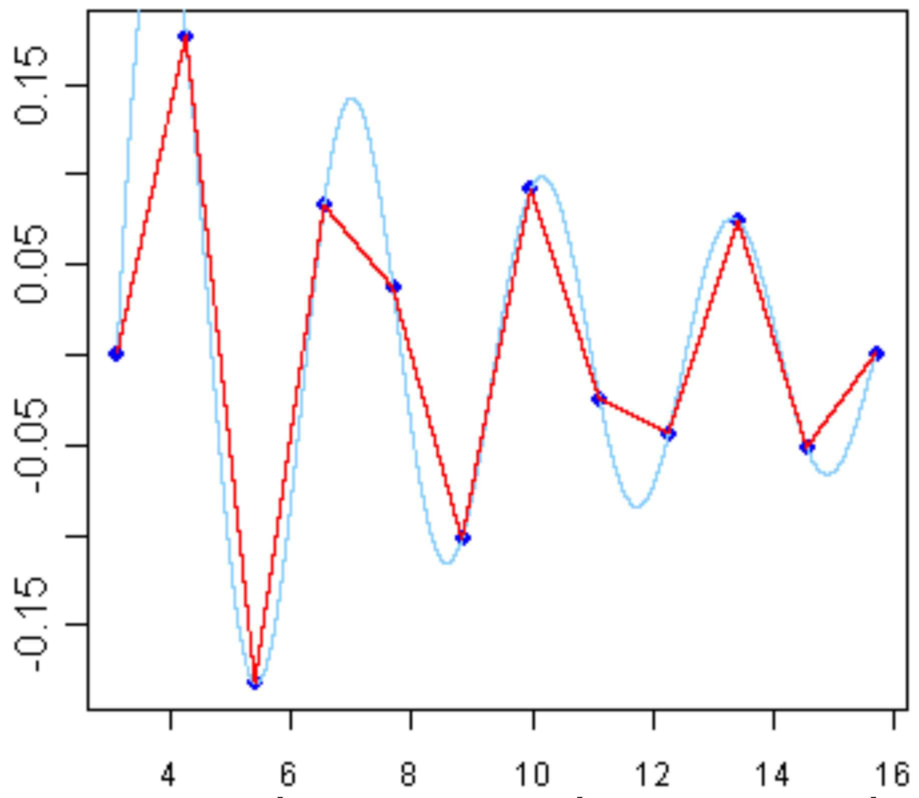


Complex Surfaces

Complex Surfaces

Piece-Wise Linear Interpolation

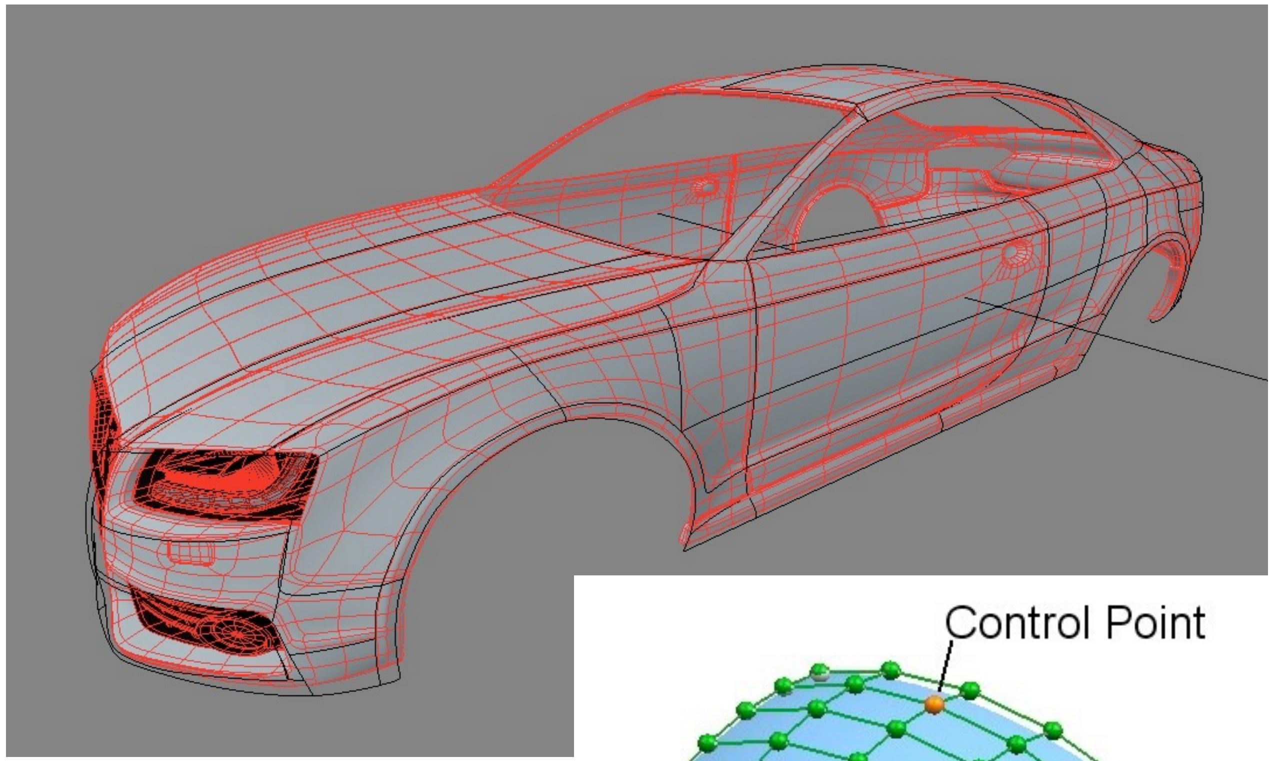
- Thus far, we used linear functions for interpolation!
- More complex geometry was stitched from multiple pieces (e.g. cells, such as cubes, triangles, line segments, etc.) that are all interpolated independently (piece-wise linear interpolation)
- Piece-wise linear interpolation has limited potential for describing complex surfaces, since they would have to be composed from many small patches/cells to avoid visible discontinuities at the edges
- Let‘s try to do a better job
- We start with curves and extend this to surfaces

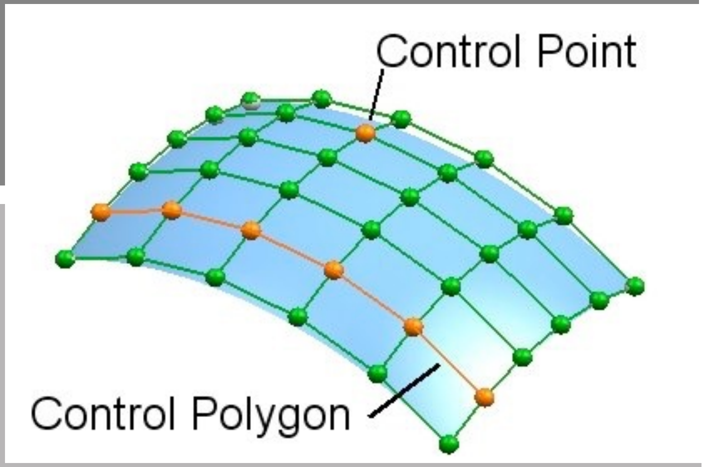


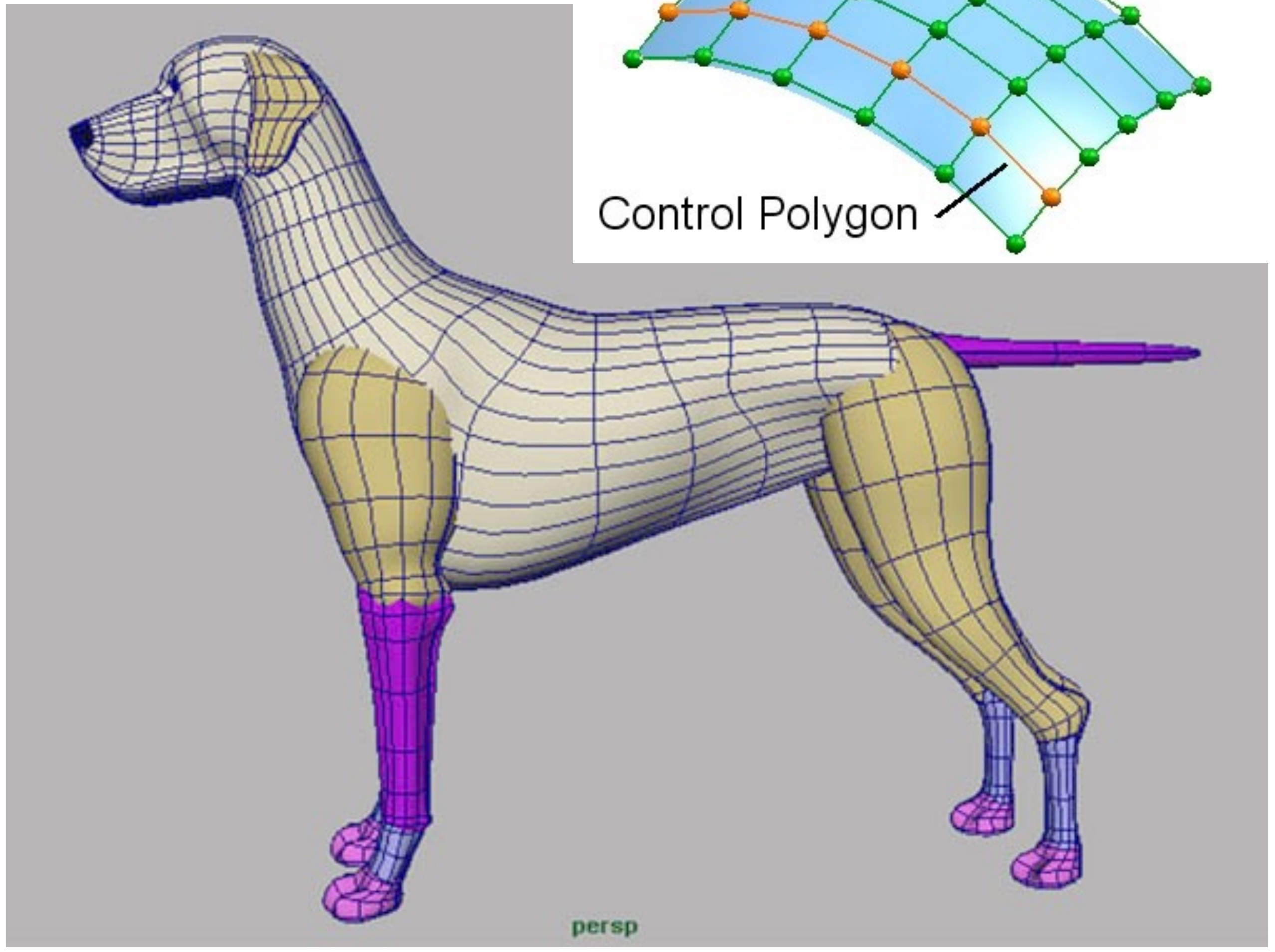
piece-wise linear vs. non-linear interpolation of a function

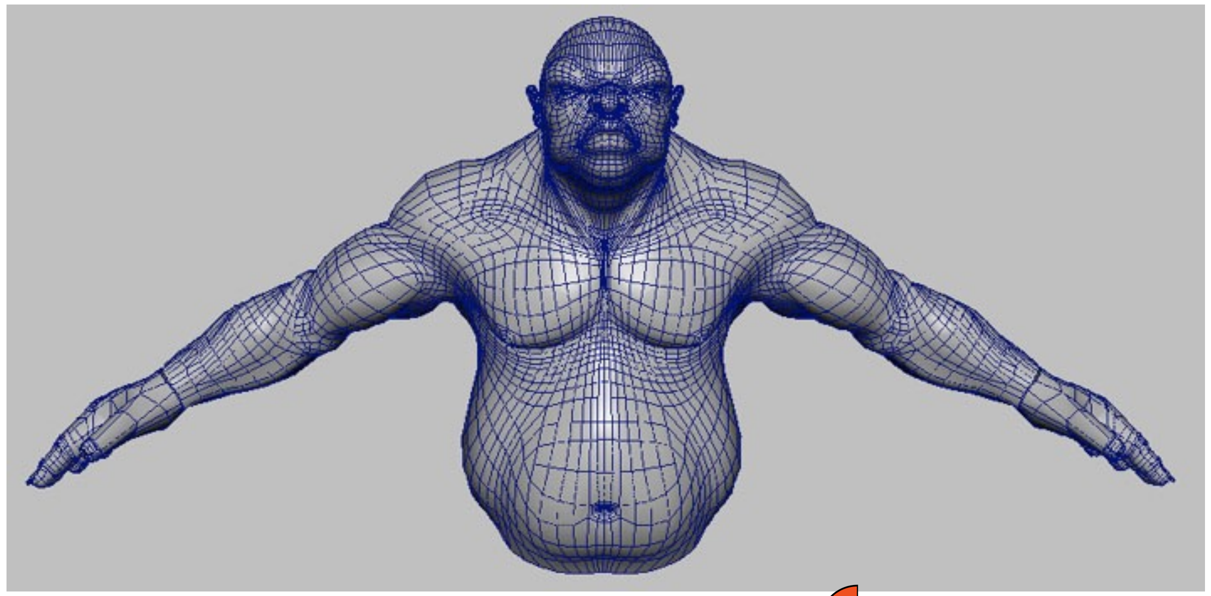
Splines

- Complicated surfaces cannot be modeled and animated by putting triangles together manually
- Instead, control points and/or control polygons are defined and intermediate surfaces portions are computed through interpolation
- This does not only apply to surfaces, but also to curves
- In general, such interpolated curves are called splines (and surfaces are called spline surfaces)
- Let‘s see how this works



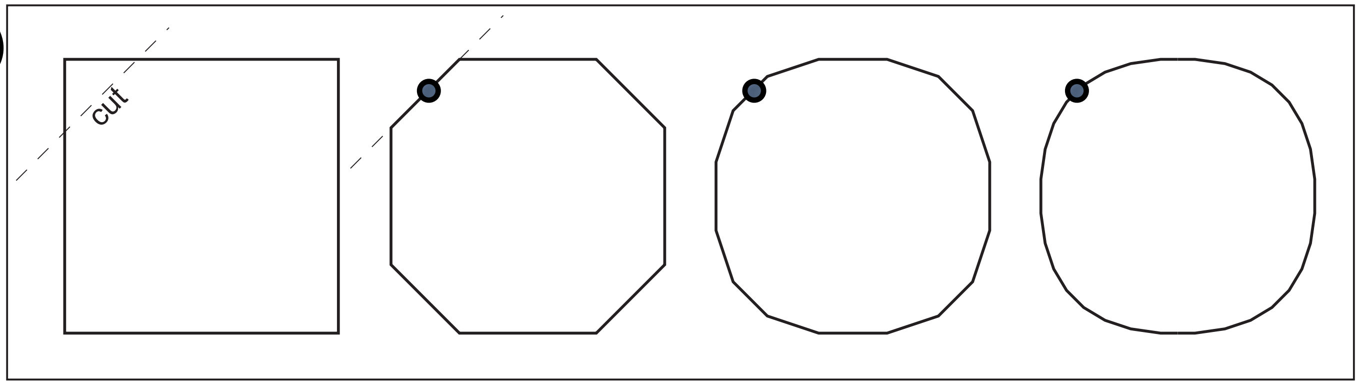






Quadratic Bézier Curves

- Let‘s assume we start with a rectangle (having four control points)
- Then we recursively cut away corners (this is referred to as a form of sub-division)
- We end up at a smooth curve, which is called limited curve
- Interesting to see is, that each midpoint of a new cutting edge already lies on the limited curve (they are never moved)!

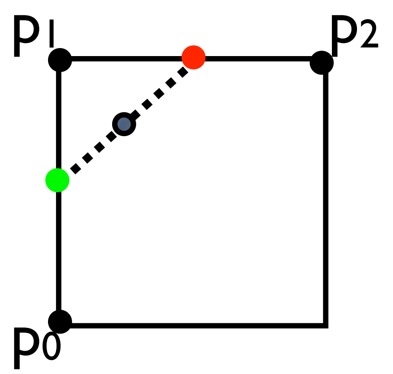


subsequent corner cutting results in limited curve

```
subdivide (p0,p1,p2)
    p01=(p0+p1)/2
    p12=(p1+p2)/2
    pm=(p01+p12)/2
    subdivide(p0,p01,pm)
    subdivide(pm,p12,p2)
```

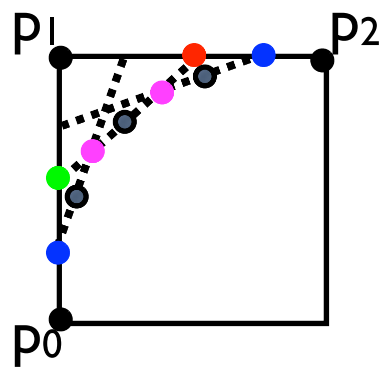
example for 2 recursions:

$$p = \frac{1}{2}\left(\frac{1}{2}p_0 + \frac{1}{2}p_1\right) + \frac{1}{2}\left(\frac{1}{2}p_1 + \frac{1}{2}p_2\right)$$



$$p = \frac{1}{4}\left(\frac{1}{4}p_0 + \frac{3}{4}p_1\right) + \frac{3}{4}\left(\frac{1}{4}p_1 + \frac{3}{4}p_2\right)$$

$$p = \frac{3}{4}\left(\frac{3}{4}p_0 + \frac{1}{4}p_1\right) + \frac{1}{4}\left(\frac{3}{4}p_1 + \frac{3}{4}p_2\right)$$



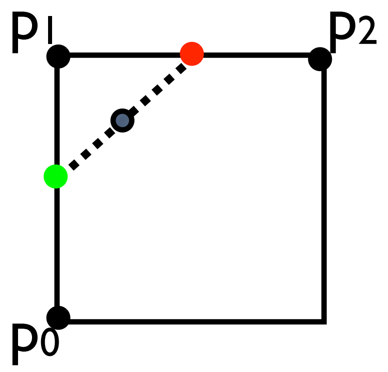
Quadratic Bézier Curves

- Let's assume we start with a rectangle (having four control points)
- Then we recursively cut away corners (this is referred to as a form of sub-division)
- We end up at a smooth curve, which is called limited curve
- Interesting to see is, that each midpoint of a new cutting edge already lies on the limited curve (they are never moved)!
- Having three control points leads to quadratic expressions when parameterizing all midpoints
- The curve that is formed by all midpoints is therefore called quadratic Bézier curve (or Bézier curve of degree two)

```
subdivide ($p_0$,$p_1$,$p_2$)
    $p_{01}=(p_0+p_1)/2$
    $p_{12}=(p_1+p_2)/2$
    $p_m=(p_{01}+p_{12})/2$
    subdivide($p_0$,$p_{01}$,$p_m$)
    subdivide($p_m$,$p_{12}$,$p_2$)
```

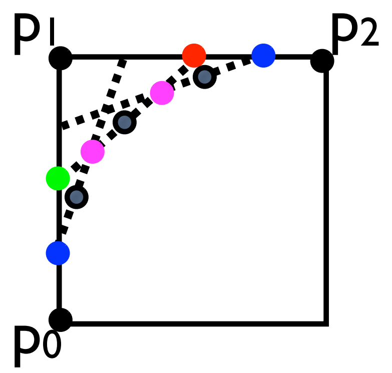
example for 2 recursions:

$$p=\frac{1}{2}\left(\frac{1}{2}p_0+\frac{1}{2}p_1\right)+\frac{1}{2}\left(\frac{1}{2}p_1+\frac{1}{2}p_2\right)$$



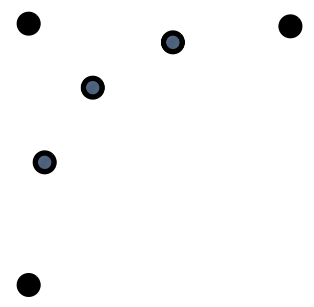
$$p=\frac{1}{4}\left(\frac{1}{4}p_0+\frac{3}{4}p_1\right)+\frac{3}{4}\left(\frac{1}{4}p_1+\frac{3}{4}p_2\right)$$

$$p=\frac{3}{4}\left(\frac{3}{4}p_0+\frac{1}{4}p_1\right)+\frac{1}{4}\left(\frac{3}{4}p_1+\frac{3}{4}p_2\right)$$



in general for t=0..1:

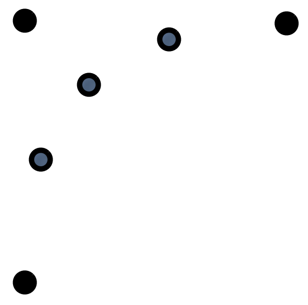
$$\begin{aligned}&p(t)=\\&(1-t)((1-t)p_0+tp_1)+\\&t((1-t)p_1+tp_2)=\\&(1-t)^2p_0+2(1-t)tp_1+t^2p_2\end{aligned}$$



Quadratic Bézier Curves

- Let's assume we start with a rectangle (having four control points)
- Then we recursively cut away corners (this is referred to as a form of sub-division)
- We end up at a smooth curve, which is called limited curve
- Interesting to see is, that each midpoint of a new cutting edge already lies on the limited curve (they are never moved)!
- Having three control points leads to quadratic expressions when parameterizing all midpoints
- The curve that is formed by all midpoints is therefore called quadratic Bézier curve (or Bézier curve of degree two)
- Thus, p(t) is the weighted average of the control points p_i (they are blended together with blending functions / base functions)

in general for t=0..1:



$$\begin{aligned}
&p(t) = \\
&(1-t)((1-t)p_0 + tp_1) + \\
&t((1-t)p_1 + tp_2) = \\
&\boxed{(1-t)^2}p_0 + \boxed{2(1-t)t}p_1 + \boxed{t^2}p_2
\end{aligned}$$

blending of control points:

$$p(t) = \sum_{i=0}^{N-1} w_i(t) p_i$$

blending functions for different control points:

$$w_0(t) = (1-t)^2$$
$$w_1(t) = 2(1-t)t$$
$$w_2(t) = t^2$$

Note, that for each parameter t, the sum of all w_i equals 1, with maximum influence at t=0,1/2,1!

Higher-Order Bézier Curves

- Extending this to four control points is straight forward (following the same procedure)
- These curves are then called cubic Bézier curves
- The subdivision idea can be generalized to any number of N control points
- The degree of the corresponding blending functions will then be N-1
- The coefficients all have the same pattern of binomial coefficients times $(1-t)^{N-i-1}t_i$
- The recursive form of blending functions is easy to implement
- Bézier curves are special forms of splines (sometimes called Bézier splines)

quadratic Bézier curves:

$$p(t) = (1-t)^2 p_0 + 2(1-t)tp_1 + t^2 p_2$$

cubic Bézier curves:

$$p(t) = (1-t)^3 p_0 + 3(1-t)^2 tp_1 + 3(1-t)t^2 p_2 + t^3 p_3$$

quadratic Bézier curves:

$$w_0(t) = (1-t)^2$$
$$w_1(t) = 2(1-t)t$$
$$w_2(t) = t^2$$

weights are normalized with maximum influence at t=0,1/2,1

cubic Bézier curves:

$$w_0(t) = (1-t)^3$$
$$w_1(t) = 3(1-t)^2 t$$
$$w_2(t) = 3(1-t)t^2$$
$$w_3(t) = t^3$$

weights are normalized with maximum influence at t=0,1/3,2/3,1

in general for t=0..1 and N control points:

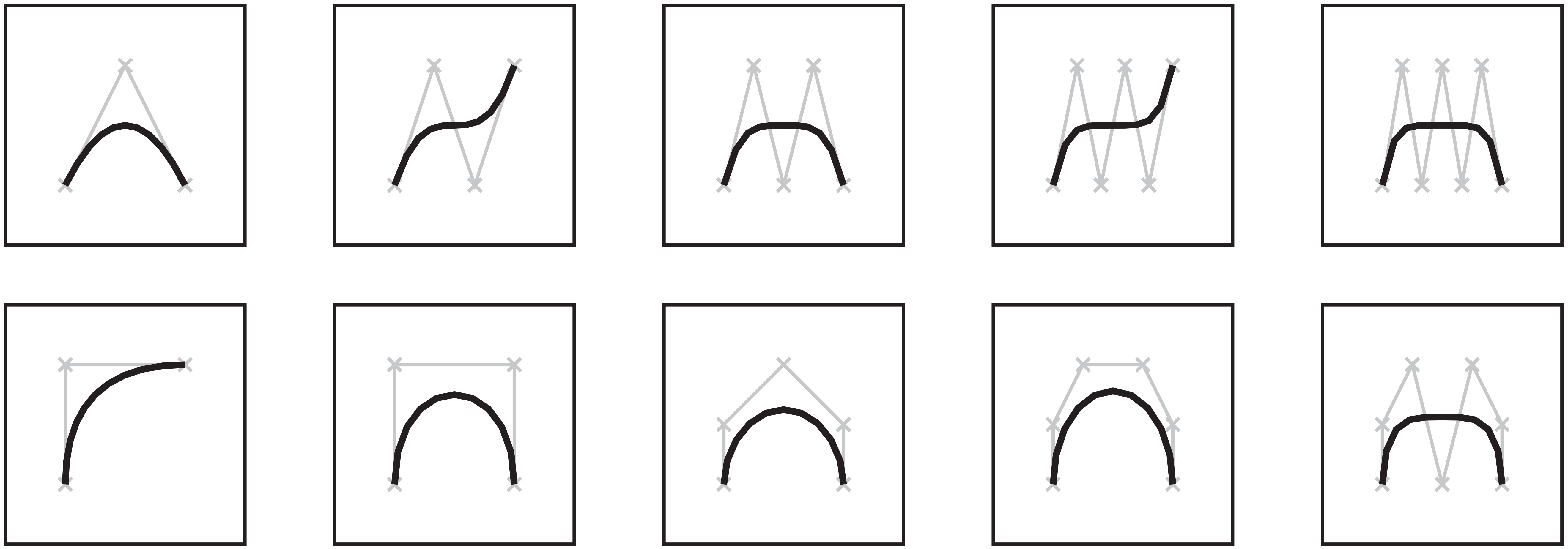
$$w_i^N(t) = \frac{(N-1)!}{i!(N-1-i)!}(1-t)^{N-1-i}t^i$$

$$p(t) = \sum_{i=0}^{N-1} w_i(t)p_i$$

$$w_i^N(t) = (1-t)w_i^{N-1} + tw_{i-1}^{N-1}$$

recursive form (easy to implement)

Higher-Order Bézier Curves



- Degree of Bézier curve is one less than number of control points
- All control points affect all parts of the curve, except the endpoints
- The set of line segments connecting the control points is called control polygon
- The beginning and the end of the curve are tangent to control polygon
- If control points are 3D, we get space curves
- How can we extend this idea to surfaces?

Bézier Surfaces

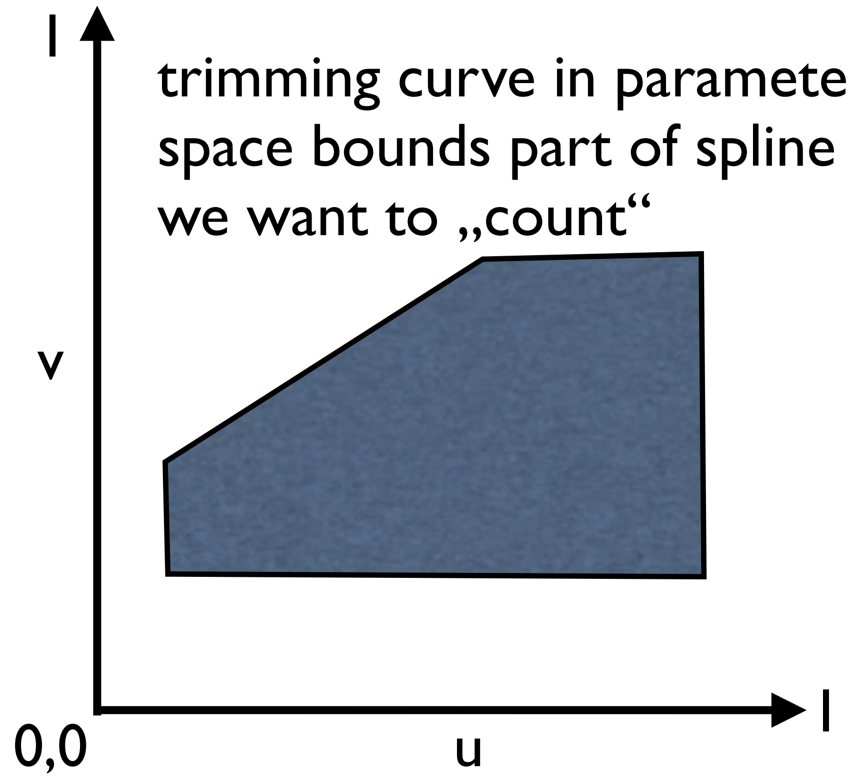
- Extending this to surfaces is straight forward (following the same procedure)
- We have 3D control points in two dimensions over a surface
- One key observation is that the blending functions become 2D (u,v parameters), and that these 2D blending functions can be formed by multiplying two 1D blending functions (one in u, and one in v)
- Such Bézier surfaces work fine for rectangular surfaces - but this is not always the case
- We can apply the same principle also to surfaces that have a non-rectangular base
- This can be done by constraining the parameter space u,v to a particular surface base using a pseudo-polygon in u,v
- Parameters outside this polygon are not considered and don‘t influence the surface
- This is called trimming, and the polygon is called trimming curve (actually, it can be a polygon and also a spline itself)

blending of control points in 2D:

$$p(u,v)=\sum_{j=0}^{M-1}\sum_{i=0}^{N-1}\boxed{w_{ij}(u,v)}p_{ij}=\sum_{j=0}^{M-1}\sum_{i=0}^{N-1}\boxed{w_i(u)w_j(v)}p_{ij}$$

example (blending function w_{12} for 4x3 control points):

$$w_1^{N=4}(u)=3(1-u)^2u$$
$$w_2^{M=3}(v)=v^2$$
$$w_{12}^{NxM=4x3}(u,v)=3(1-u)^2uv^2$$

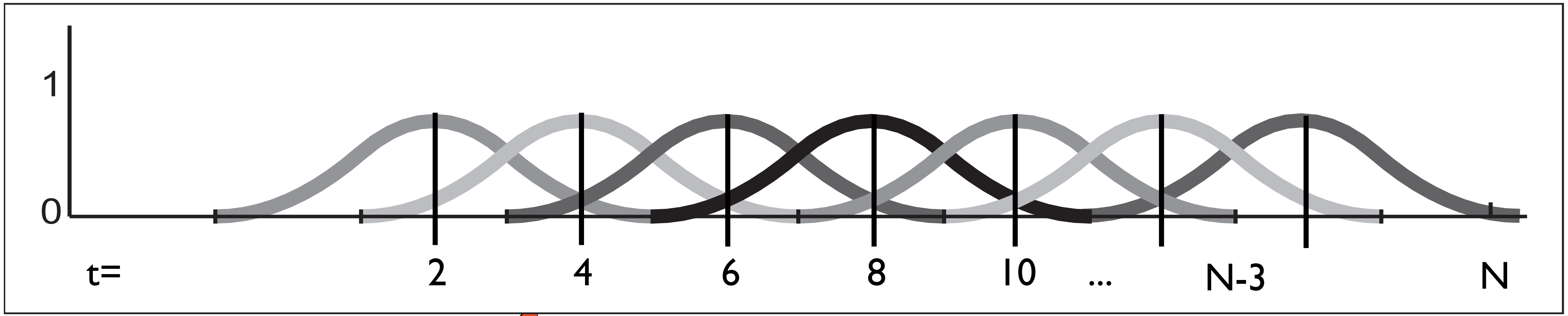

1
trimming curve in parameters space bounds part of spline surface we want to „count“
v
0,0
u
1

Uniform B-Splines

- One problem with Bézier curves is that if they have many control points, we have a very high-degree polynomial, which quickly leads to computational problems
- Another problem with Bézier curves is that every control point influences the curve, which seems to be unnecessary if local sections need to be modified
- Can we create a low-degree polynomial that uses blending functions which are limited to a small local region?
- We can create blending functions made up of pieces of polynomials (which is then called B-spline blending function, and the resulting curve is called B-spline)
- These pieces (i.e., the basis functions) are shifted along a parameter space t=3..N-3 (rather than t=0..1), in such a way that the maximal influence of each basis function corresponds with the appropriate control point
- The sum of all contributions of all (control point individual) blending functions gives the new curve points:

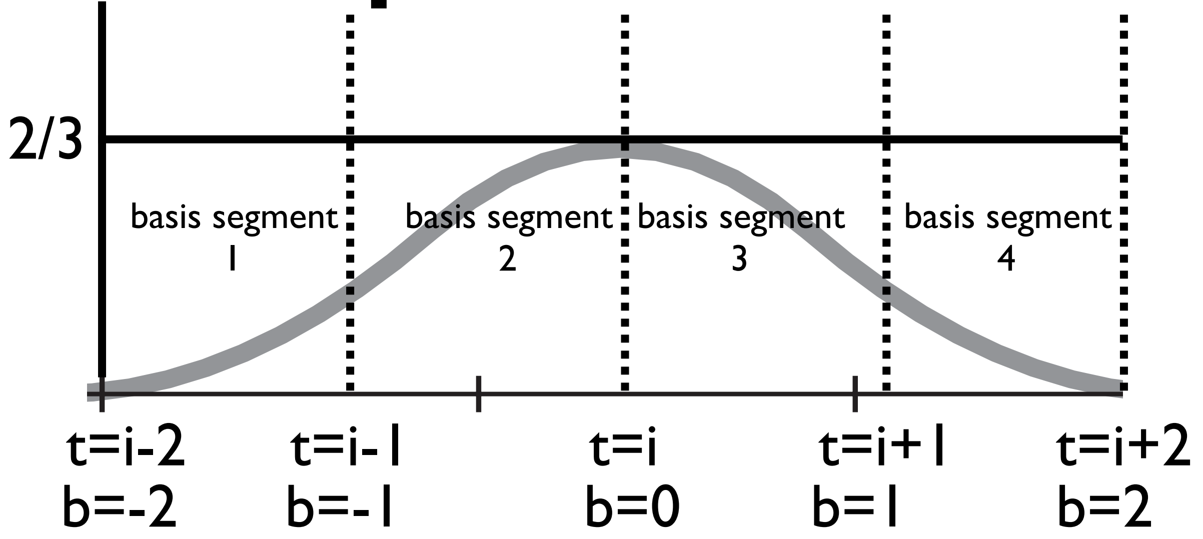
$$p(t) = \sum_{i=0}^{N-1} w(t \boxed{-i}) p_i$$

leads to the shift of the same basis function



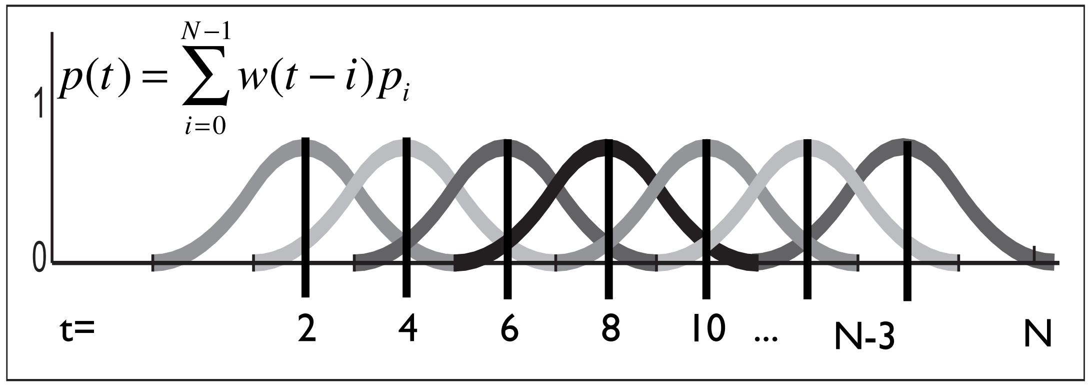
Uniform Cubic B-Splines

- In this example, the same basis function is shifted in such a way that its maximum is at t=i
- The basis function is stitched together from four basis segments
- Each basis segment has to be evaluated in the local rage of 0..1 (thus we have to normalize t with respect to the corresponding segment)
- The sum of all blending functions at every t is 1, again
- On a global basis, t's range is 3..N-3 (parameters 0..2, and N..N-2 are not considered, since there are not enough points for the blending functions to sum to 1 in those regions)



basis segment | normalizing to 0..1 | local range

$$w(b)=\begin{cases} \frac{1}{6}t^3, t=b+2 & -2<b\le -1 \\ \frac{1}{6}\left(-3t^3+3t^2+3t+1\right), t=b+1 & -1<b\le 0 \\ \frac{1}{6}\left(3t^3-6t^2+4\right), t=b & 0<b\le 1 \\ \frac{1}{6}\left(-t^3+3t^2-3t+1\right), t=b-1 & 1<b\le 2 \\ 0 & otherwise \end{cases}$$

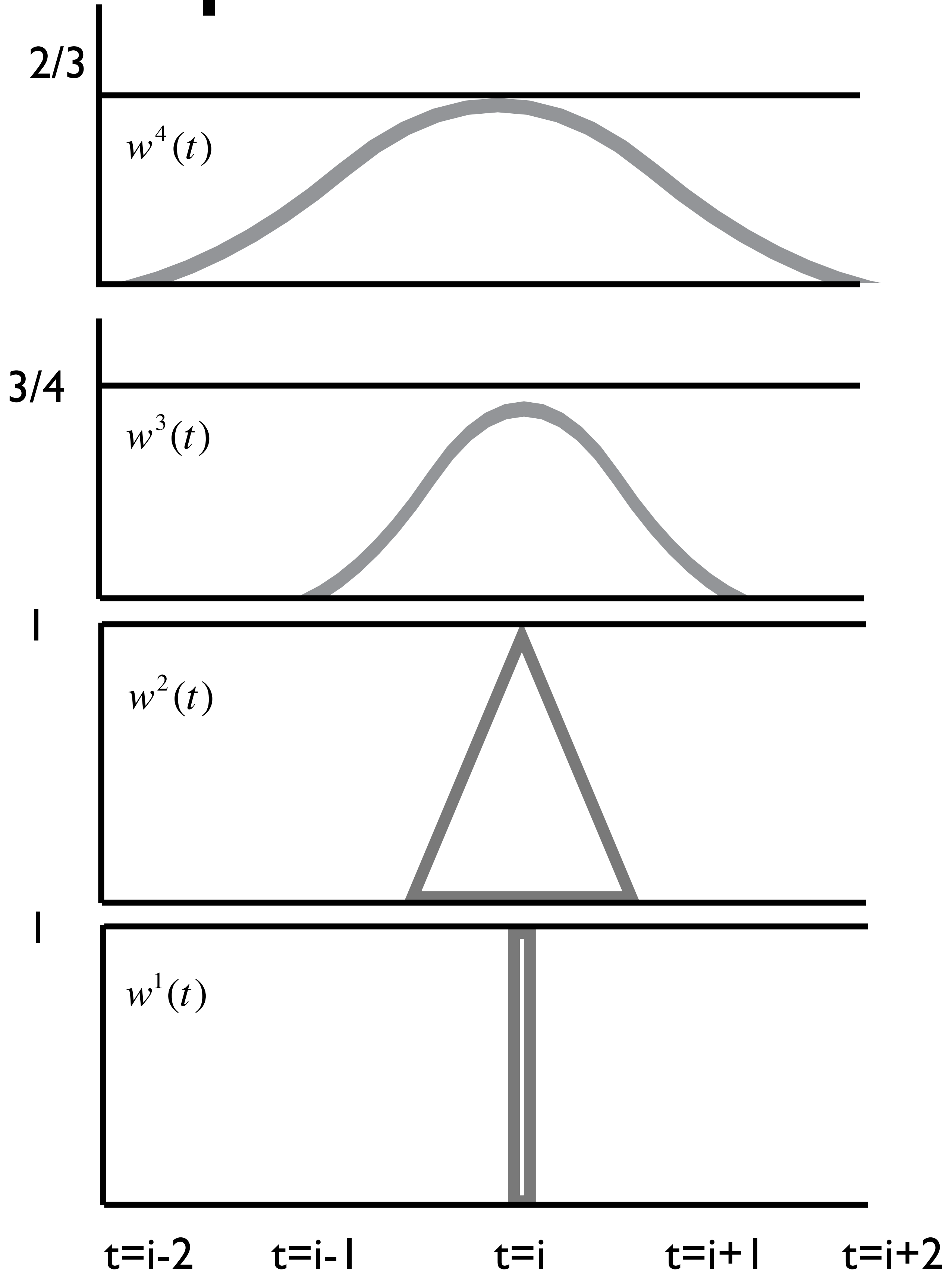


Non-Cubic B-Splines

- One control point of a cubic B-spline influences a local region of t=i-2…t=i+2 on the curve
- This region can be made smaller or larger by using lower-order or higher order blending functions
- Cubic B-spline blending functions are just one of a family of blending functions
- As for Bézier blending functions, the B-spline blending functions can be computed recursively (after mapping t to local range!):

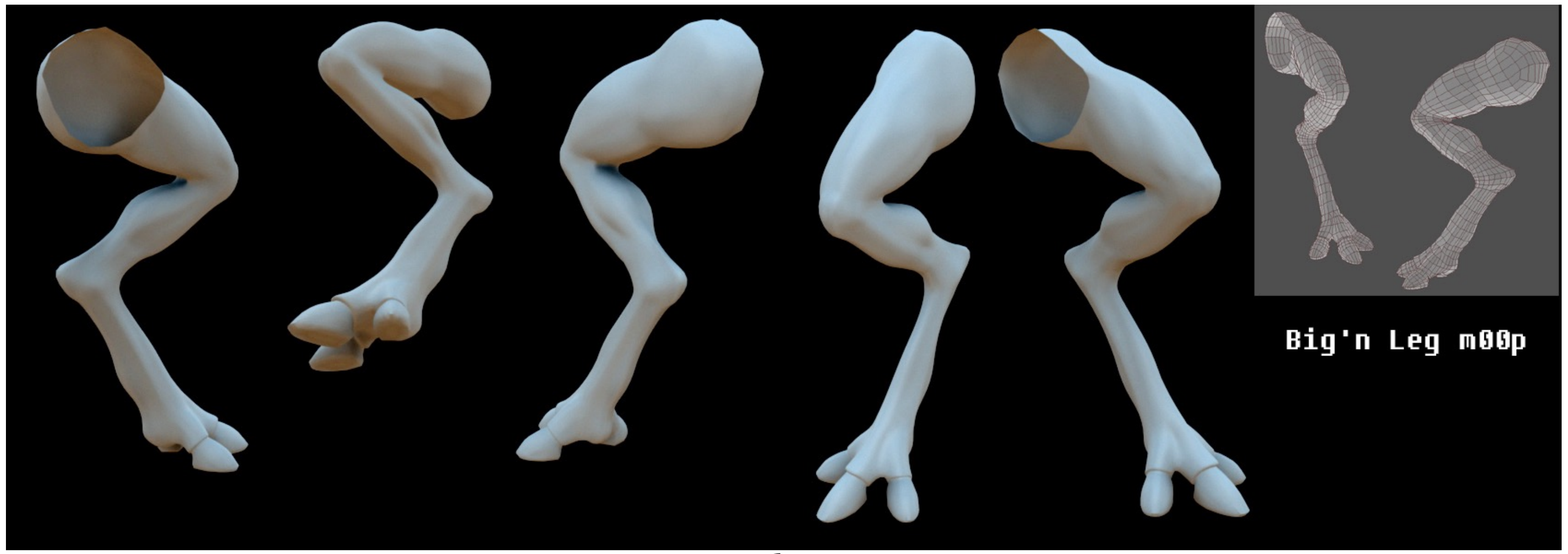
$$w^k(t) = \frac{t}{k-1} w^{k-1}(t) + \frac{k-t}{k-1} w^{k-1}(t+1)$$

$$w^1(t) = \begin{cases} 1 & 0 < t \le 1 \\ 0 & otherwise \end{cases}$$



NURBS

- In practice, a more complicated form of B-splines are used, rather than uniform B-splines
- They are called nonuniform rational B-splines (NURBS)
- The blending function of NURBS is the ratio of two polynomials (by definition, this is a rational function)
- The advantage of (rational) NURBS compared to (polynomial) B-splines is that they can represent conics exactly (ie., they are preferred when modeling with cylinders and spheres)
- Nonuniform means that the gaps in t between the bending functions do neither have to be uniform, nor an integer (can be user defined)



NEXT ICG LAB TALK:
JUNE 14, 2016, 4:30PM

Institute of Computer Graphics

Prof. Thomas Pock

Graz University of Technology

Learning better models for computer vision

Science Park 3

Room S3 048

JKU

For more information about our talks visit
http://www.cg.jku.at/talks/invited

Course Schedule

Type	Date	Time	Room	Topic	Comment
C1	01.03.2016	13:45-15:15	HS 18	Introduction and Course Overview	Conference
C2	15.03.2016	13:45-15:15	HS 18	Transformations and Projections	Easter Break
C3	05.04.2016	13:45-15:15	HS 18	Raster Algorithms and Depth Handling	
C4	12.04.2016	13:45-15:15	HS 18	Local Shading and Illumination	
C5	19.04.2016	13:45-15:15	HS 18	Texture Mapping Basics	
C6	26.4.2016	13:45-15:15	HS 18	Advanced Texture Mapping & Graphics Pipelines	
C7	03.05.2016	13:45-15:15	HS 18	Intermediate Exam	
C8	09.05.2016	17:15-18:45	HS 18	Global Illumination I: Raytracing	
C9	10.05.2016	13:45-15:15	HS 18	Global Illumination II: Radiosity	Conference / Holiday
C10	31.05.2016	13:45-15:15	HS 18	Volume Rendering	
C11	07.06.2016	13:45-15:15	HS 18	Scientific Data Visualization	
C12	14.06.2016	13:45-15:15	HS 18	Curves and Surfaces	
C13	21.06.2016	13:45-15:15	HS 18	Basics of Animation	
C14	28.06.2016	13:45-15:15	HS 18	Final Exam	
C15	04.10.2016	13:45-15:15	TBA	Retry Exam	

Thank You!