SIGGRAPH2007

Real-Time Tessellation on GPU

Natalya Tatarchuk

Game Computing Applications Group

AMD Graphics Products Group

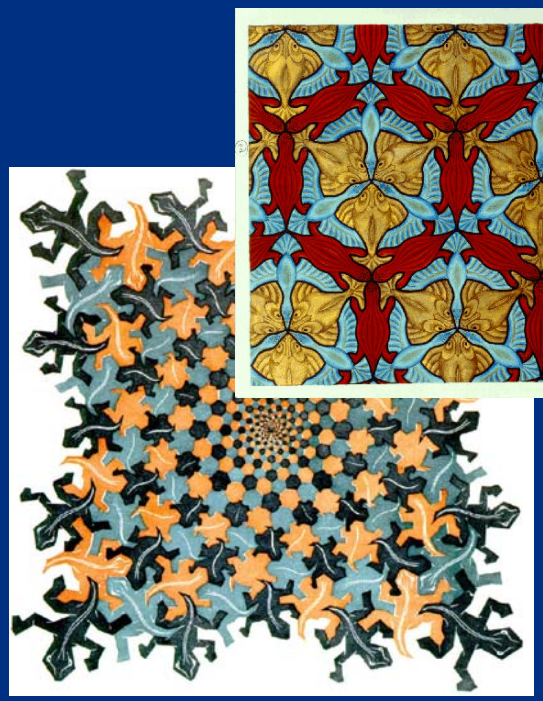
Overview

- Introductory overview of subdivision
- Tessellation on GPU
  - New pipeline
  - Character rendering
  - Animated objects
  - Terrain rendering
  - Adaptive tessellation
- Conclusions

Tessellation: An Age-Old Concept, Yet Ever Elegant

- Catmull-Clark / Doo-Sabin [1978]
- But really an enduring concept
  - *Tessella* == Mosaics
- Famously explored thoroughly by M. C. Escher
- Mathematical formulations at its roots

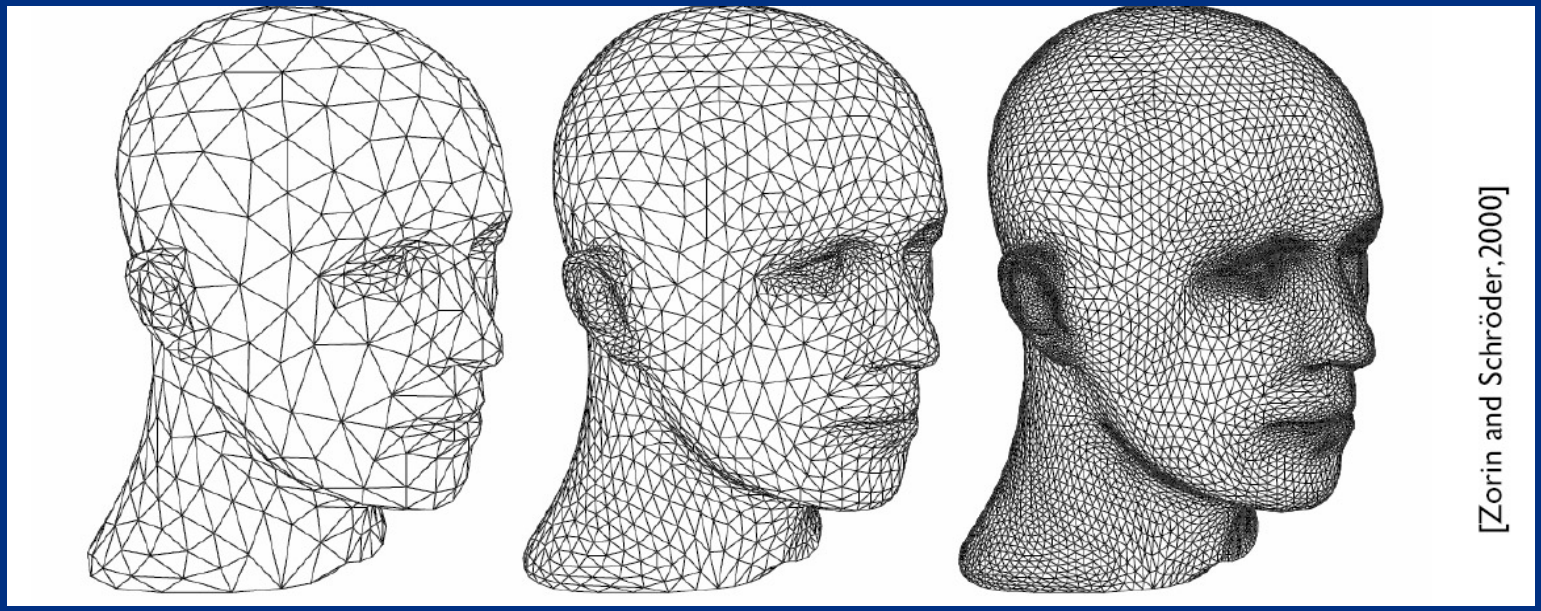
Alhambra, Granada, Spain



M.C Escher

Tessellation Process

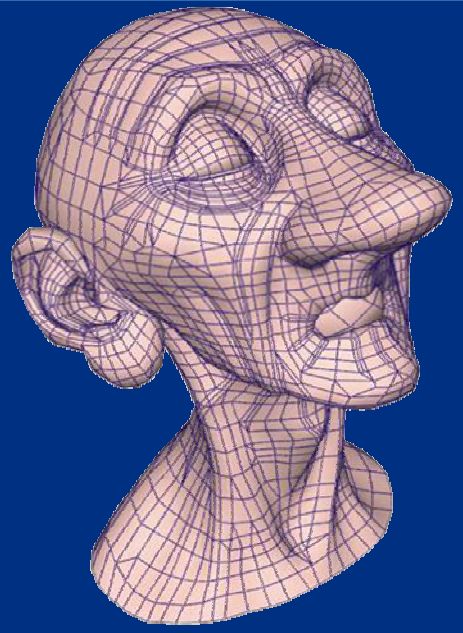
- Start with a polygon mesh
- Recursively apply subdivision rule



Subdivision Surface Types

Many different types of higher order surfaces

- Bezier
- N-Patches
- B-Spline, NURBs, NUBs
- Loop, Catmull-Clark and other subdivision surfaces



Catmull-Clark surfaces for a cinematic character [DeRose98]

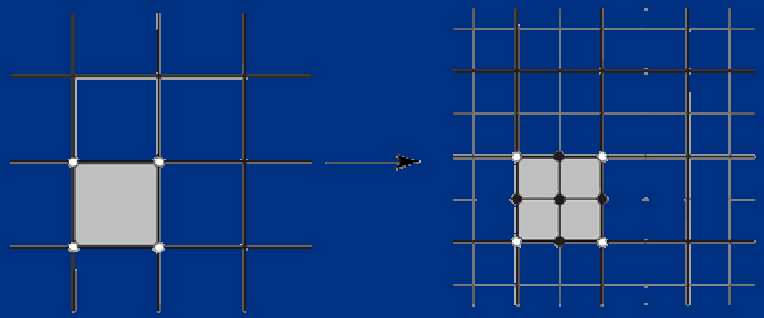
There isn't just one way to skin a cat!

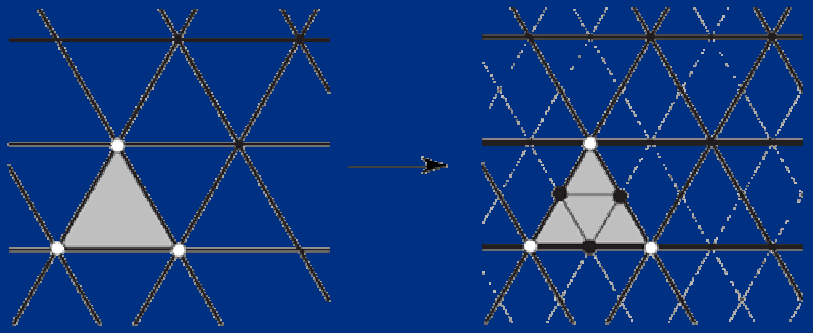
- Everything needs to be customized depending on the task

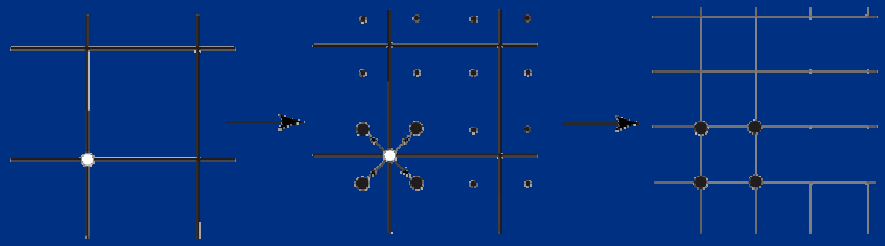
Refinement Rule: Face or Vertex Splits

Face Split

Vertex Split

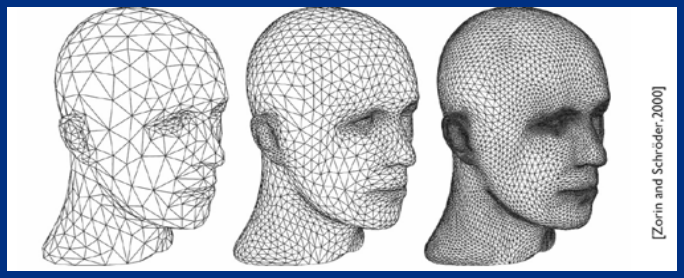


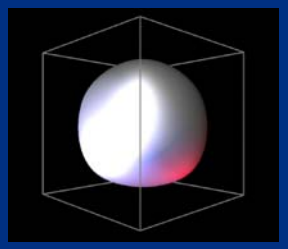




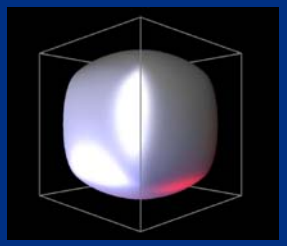
Types of Subdivision

- Interpolating
  - The original control points remain the subdivided mesh
- Approximating
  - New points are inserted and old points are moved in each step of subdivision (e.g. Loop, Catmull-Clark)
- Many methods for deciding where to place the new points
  - Resulting mesh's shape and smoothness depends on the rule

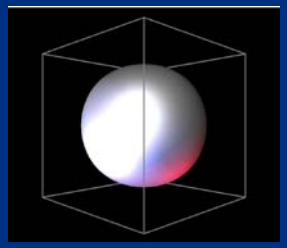




Loop



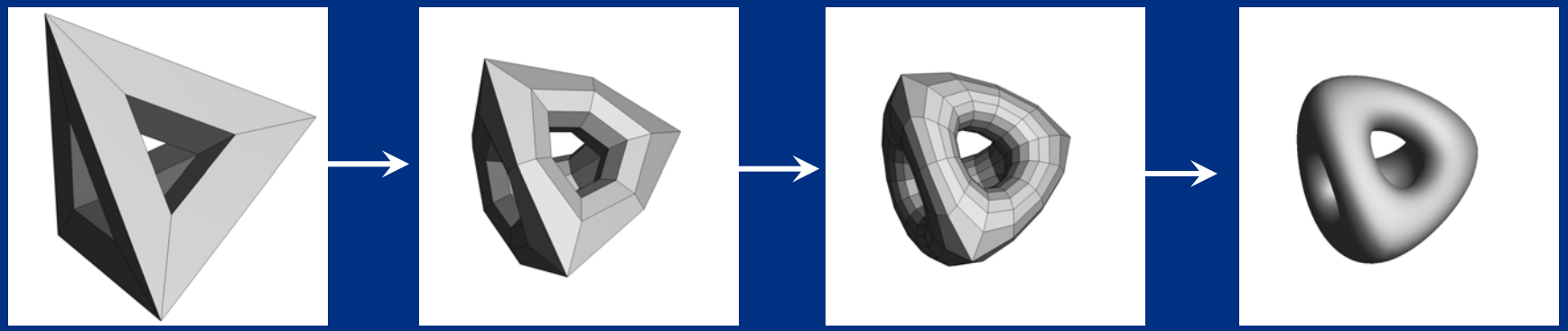
Catmull-Clark



Doo-Sabin

Subdivision Surface

– *Limit surface* - the final, super-high-resolution surface
  - Result of the subdivision process
– Dual nature
  - Can behave as if they consist purely of patches, or, alternatively, of polygons

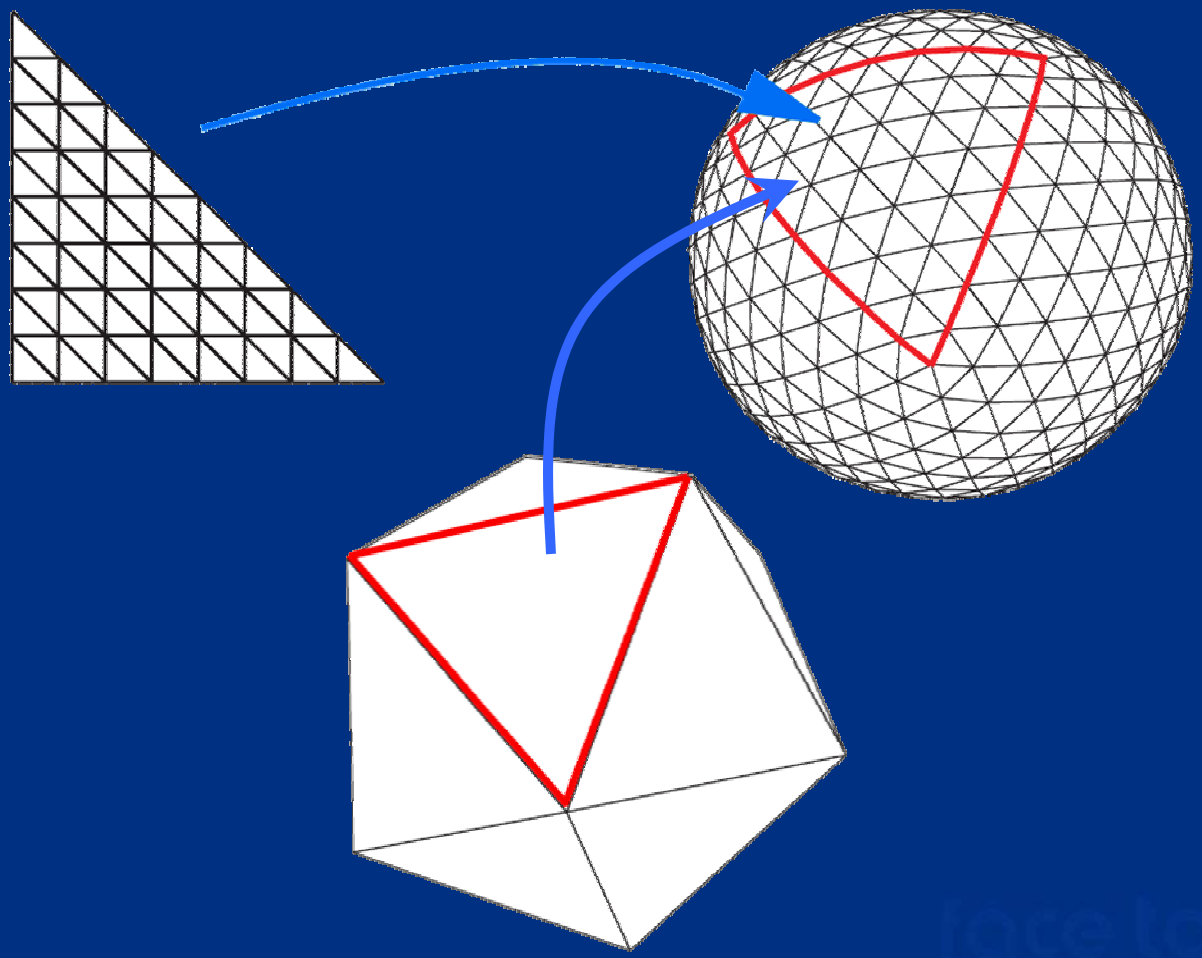


[DeRose98]

Subdivision Surface Domain

- Subdivision results in an increasing number of mesh faces and vertices
- Need to define limit surfaces precisely
- Represent as functions defined on some parametric domain ($\mathbf{R}^3$)
- Initial control mesh == *domain of the surface*

Domain Parametrization



Meshes are Expensive Beasts!

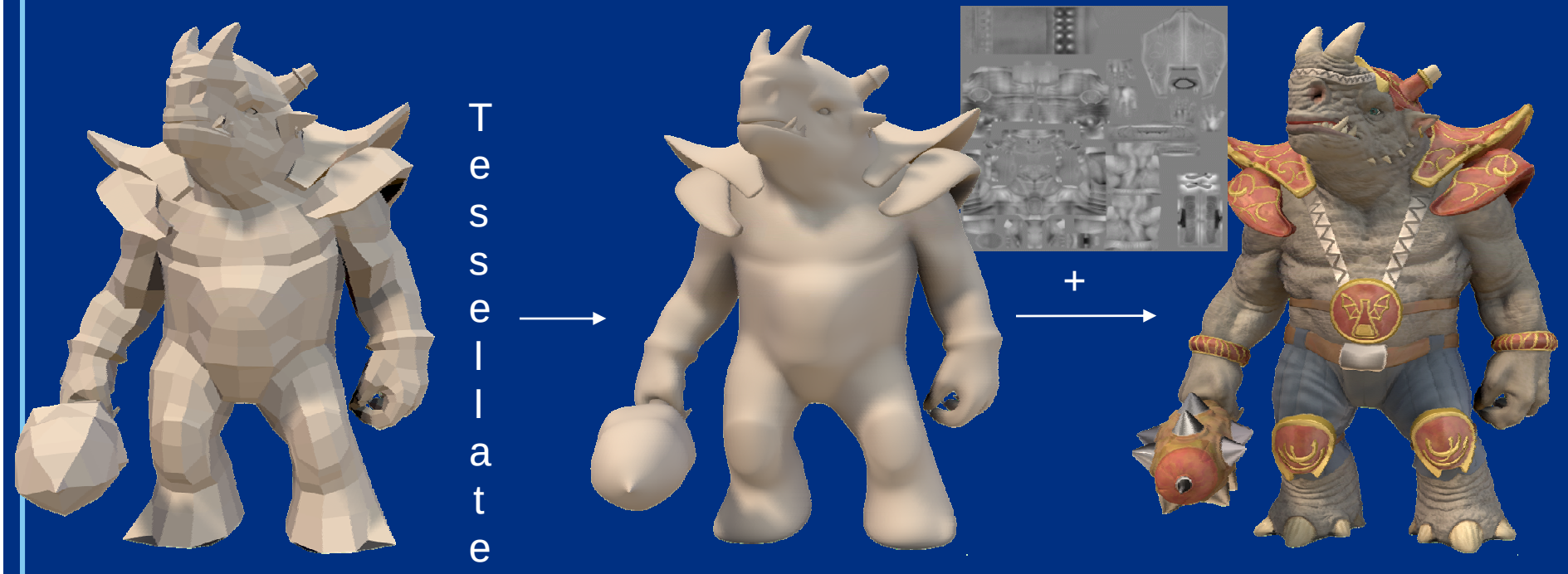
- Want to express more and more details
  - Complex, rich worlds of recent games require detailed characters
  - Shading has evolved tremendously in the recent years
- Meshes are a pricey representation
  - Need to store a lot of data (positions, *uv*s, animation)
- Good news: novel GPUs' unified shader architecture has much more efficient geometric processing
  - But memory storage and fetch bandwidth is still a big concern
  - Especially for animation
  - Large meshes cause less vertex cache reuse

Displaced Subdivision Surfaces

- Introduced in 2000 by Lee et al
- Displacement maps are derived from the limit surface
- Typical limit surface: 200-500M triangles
  - Not used for directly for rendering even offline
- And what about games?
  - Typical character - 2K polygons

Images © Fantasy Lab and Wizards of the Coast

Displaced Subdivision Surfaces



Images © Fantasy Lab and Wizards of the Coast

Advantages of Tessellation

- Compression of vertex data
- Scalability with a knob
- Stable performance
- Amplification of animation data / morph targets / deformation models
- Allows providing data to GPU at coarser resolution while rendering at high resolution
- Displacement mapped surfaces become first class citizens

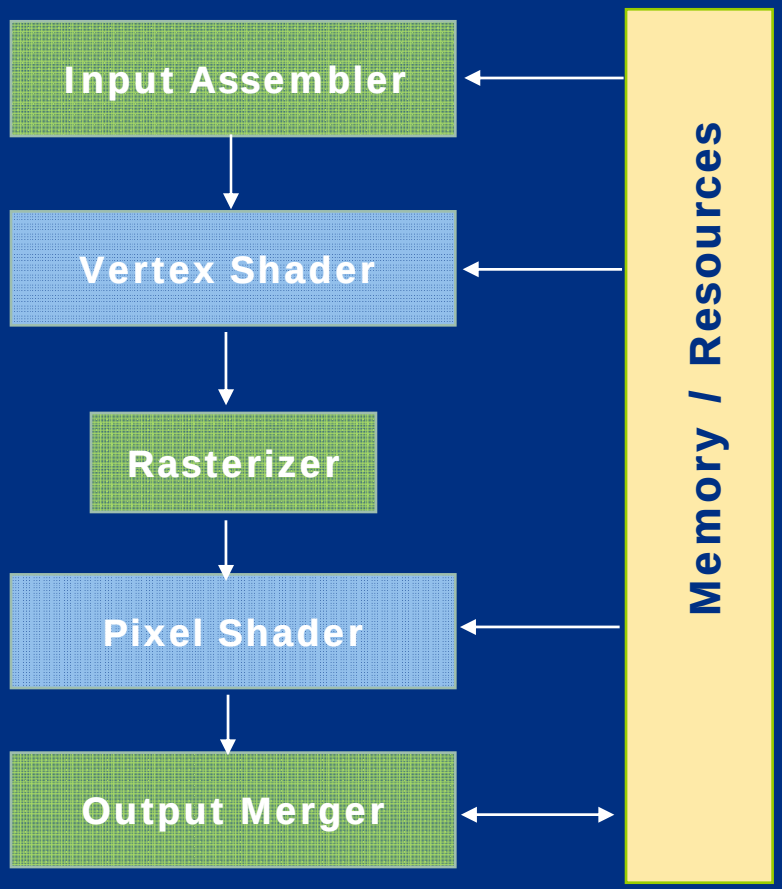
Bridging the Gap on Visual Quality for Games

- Inflection points for games: **content generation and rendering**
  - Character modeling with detail and deformation
- Transition to deformable subdivision surfaces with displacement
  - Much higher quality

Per-Frame	Shrek	Typical Game
Content size	100 M polygons	200-500K polygons
Animation quality	350 bones skinned on the CPU	32-40 bones skinning on GPU
Rendering time	8000 sec a frame on a Pentium IV	0.015 sec or less a frame

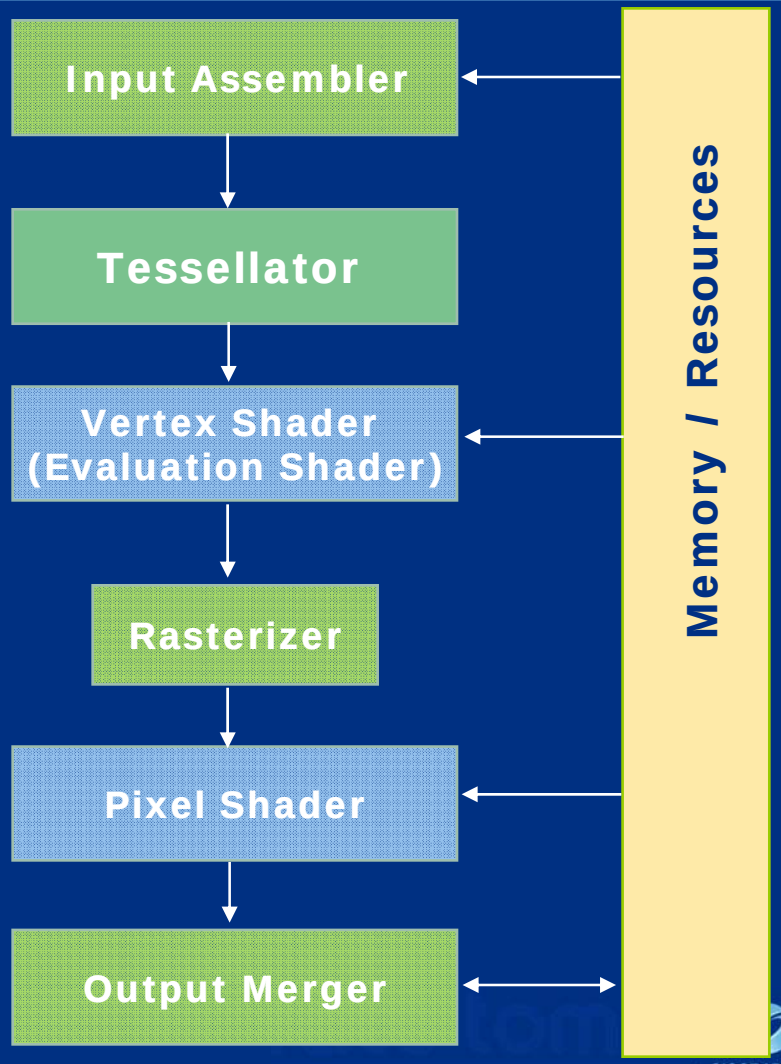
A frame from the Shrek [Yee04, PDI/Dreamworks]

Rendering Pipeline Before Tessellation



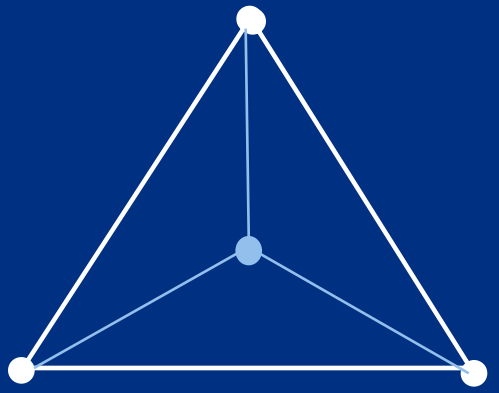
New GPU Tessellation Pipeline

- Fast tessellator hardware
  - Generates topology from a primitive or a patch
  - Parallel evaluation of new vertices and fast (u,v) generation
  - Up to 15X tessellation
- No new API shader necessary
  - Fully custom evaluation vertex shaders
- Supports various surface formulations
- Supported by the entire ATI Radeon HD 2000 Series

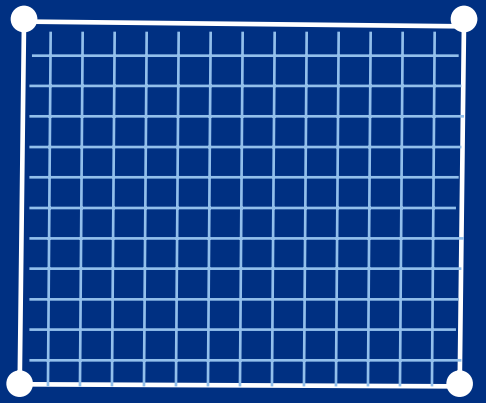


Primitives Supported by the Tessellator

- Triangles and Tri-patches



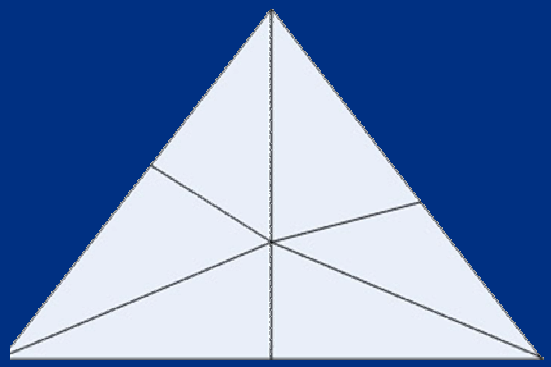
- Quads and Quad Patches



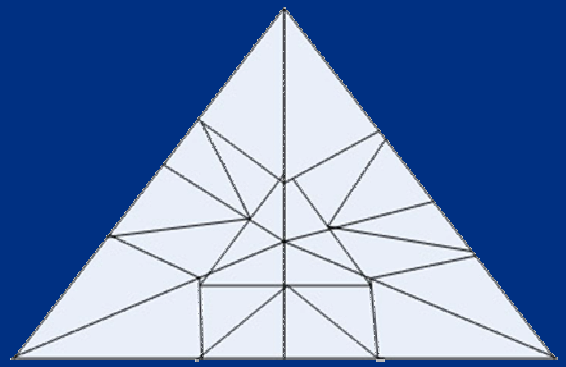
- Lines and Line Patches

Continuous Tessellation

- Specify floating point tessellation level per-draw call
- Eliminates popping as vertices are added through tessellation
- Watertight tessellation

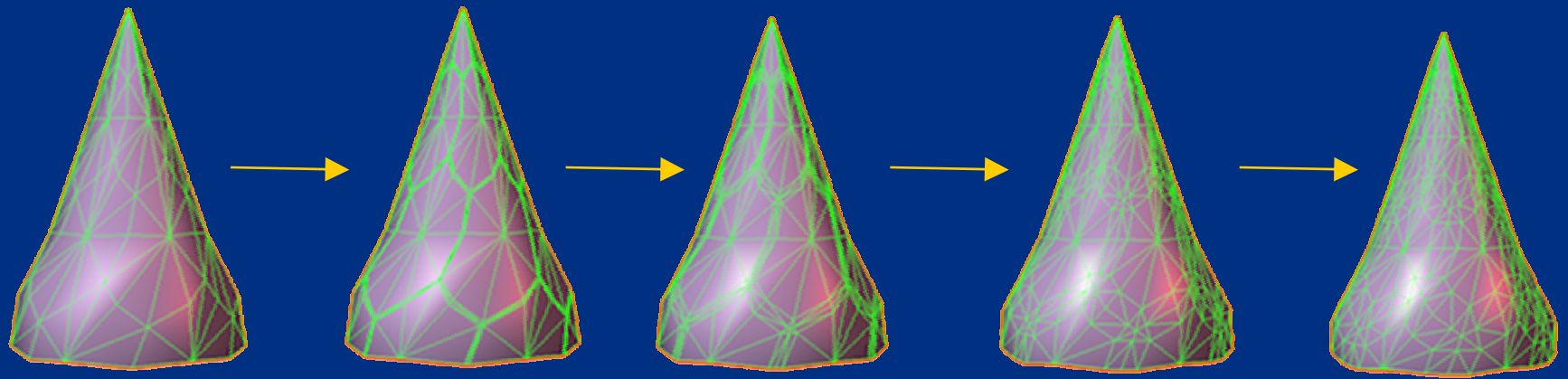


Level 1.0



Level 2.0

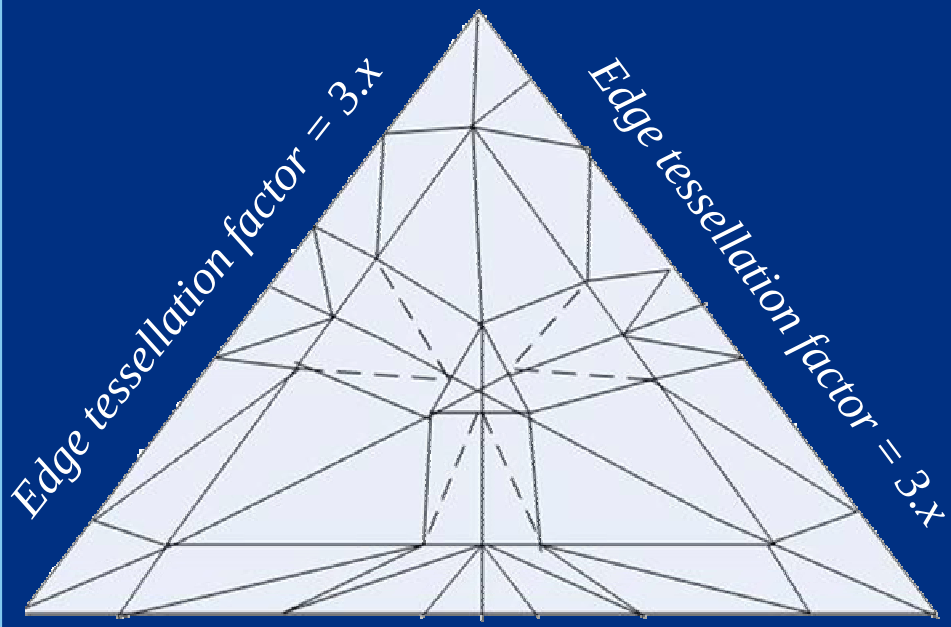
Continuous Tessellation: Increasing Levels

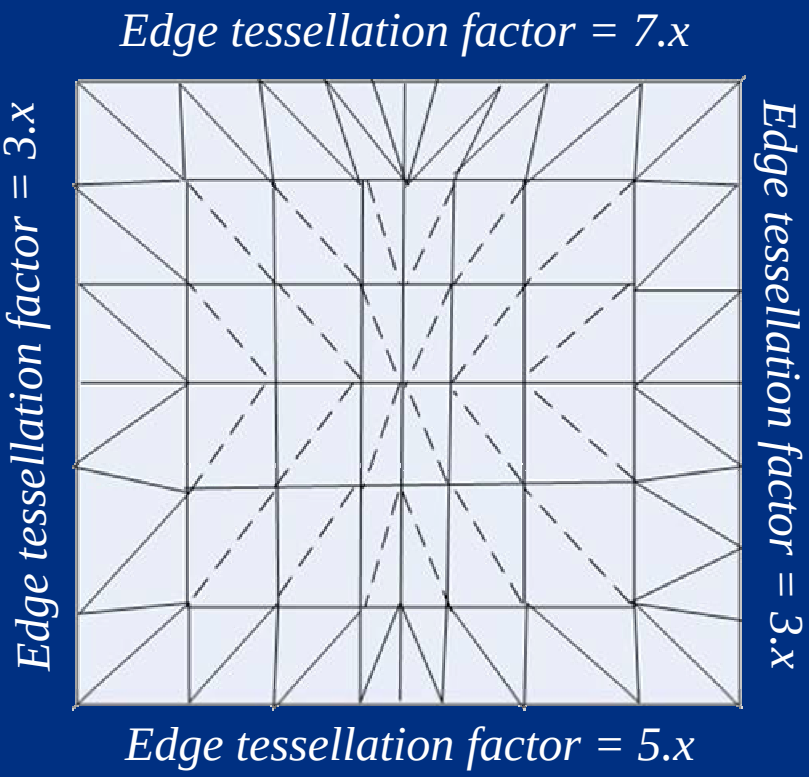


Level = 1.0 Level = 1.1 Level = 1.3 Level = 1.7 Level = 2.0

Dynamic Adaptive Tessellation

- Tessellation level is specified per edge



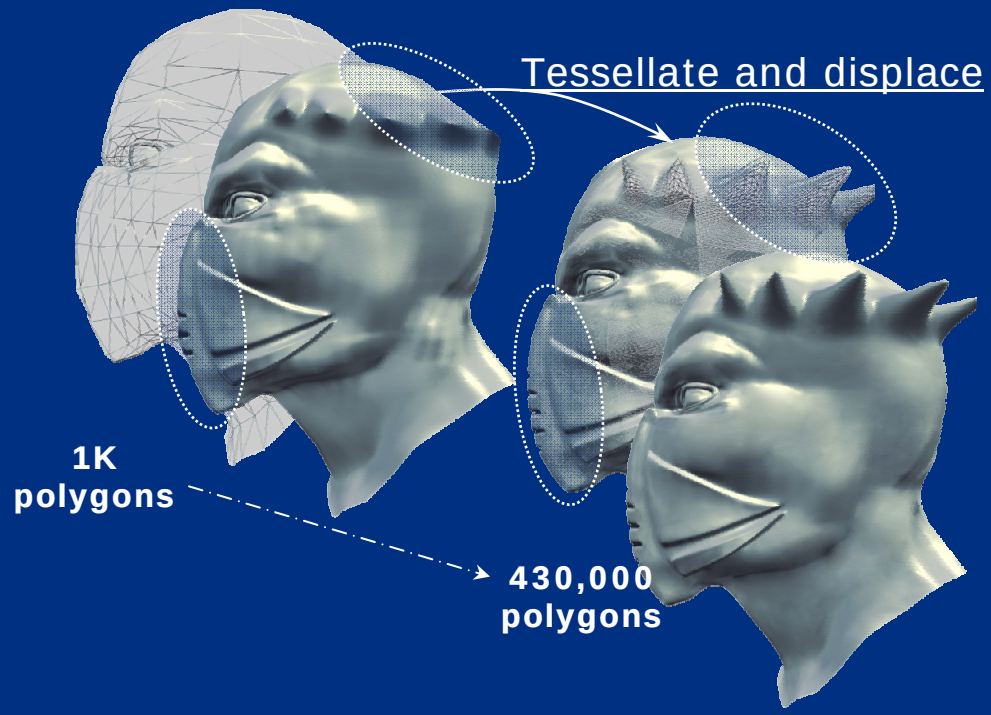


Rendering with Adaptive Tessellation

- Control min / max tessellation levels per object
  - Clamps per-edge tessellation factors to this ranges
- Similar to XDK, render with non-indexed primitives
  - Necessary in order to compute and store per-edge factors
  - Not a big impact on performance
    - All expensive vertex operations should be done in pre-pass (transforms / animation, etc)
    - Also will render the low-resolution control cage

Tessellated Characters Rendering

- Get up close to a character and see a lot of details
- Level-of-detail management
- Start by rendering low-resolution character
  - That's the *control cage*
  - Tessellate and displace for finer details
- Uses the same art assets
  - Displacement map



Example: A Gothic Ninja

Example: Simple Evaluation Shader

```
float4x4 mWVP;
float4x4 mMW;
float     fDisplacementScale;
float     fDisplacementBias;
sampler2D sDisplacement;
struct VsInput
{
   float3 vBarycentric: BLENDWEIGHT0;
   // Superprim vertex 0:
   float4 vPositionVert0 : POSITION0;
   float2 vTexCoordVert0 : TEXCOORD0;
   float3 vNormalVert0   : NORMAL0;
   // Superprim vertex 1:
   float4 vPositionVert1 : POSITION4;
   float2 vTexCoordVert1 : TEXCOORD4;
   float3 vNormalVert1   : NORMAL4;

   // Superprim vertex 2:
   float4 vPositionVert2 : POSITION8;
   float2 vTexCoordVert2 : TEXCOORD8;
   float3 vNormalVert2   : NORMAL8;
};
struct VsOutput
{
   float4 vPosCS      : POSITION;
   float2 vTexCoord   : TEXCOORD0;
   float3 vNormalWS   : TEXCOORD1;
   float3 vPositionWS : TEXCOORD2;
};
```

Example: Simple Evaluation Shader (cont.)

```
VsOutput VS( VsInput i )

{

   VsOutput o;

   // Compute new position based on the
    // barycentric coordinates:

   float3 vPosTessOS =
    i.vPositionVert0.xyz * i.vBarycentric.x
    + i.vPositionVert1.xyz *
    i.vBarycentric.y +
    i.vPositionVert2.xyz * i.vBarycentric.z;



    // Output world-space position:
    o.vPositionWS = vPosTessOS;

    // Compute new tangent space basis
    // vectors for the tessellated vertex:

    o.vNormalWS = i.vNormalVert0.xyz *
    i.vBarycentric.x + i.vNormalVert1.xyz *
    i.vBarycentric.y +
    i.vNormalVert2.xyz * i.vBarycentric.z;
```

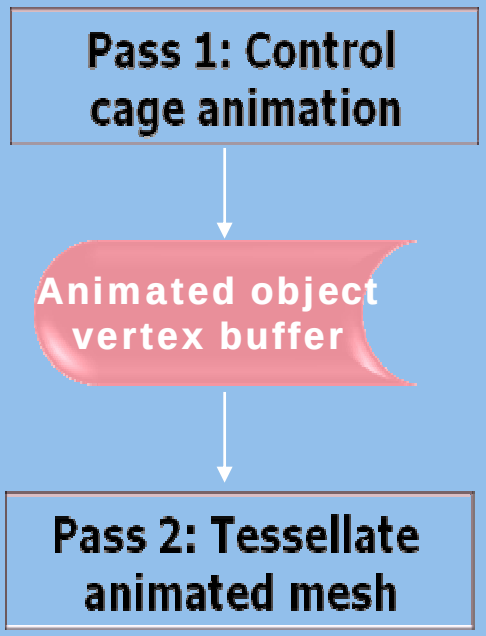
```
// Compute new texture coordinates based
  // on the barycentric coordinates:

o.vTexCoord = i.vTexCoordVert0.xy *
  i.vBarycentric.x + i.vTexCoordVert1.xy *
  i.vBarycentric.y +
  i.vTexCoordVert2.xy * i.vBarycentric.z;

// Transform position to screen-space:
  o.vPosCS = mul( mWVP,
  float4( vPosTessOS, 1.0 ) );
```

The Time-Consuming Art of Authoring Animation

- Our ultimate goal: display highly detailed, dynamic characters / objects
- Creating animation data for large, high resolution models is difficult
  - Tremendous amount of animation data – storage costs
  - High cost of vertex transform
  - Authoring that much animation is painful and time-consuming
- Animation on the control cage is the preferred method

Higher Quality Animation at Lower Cost with Tessellation

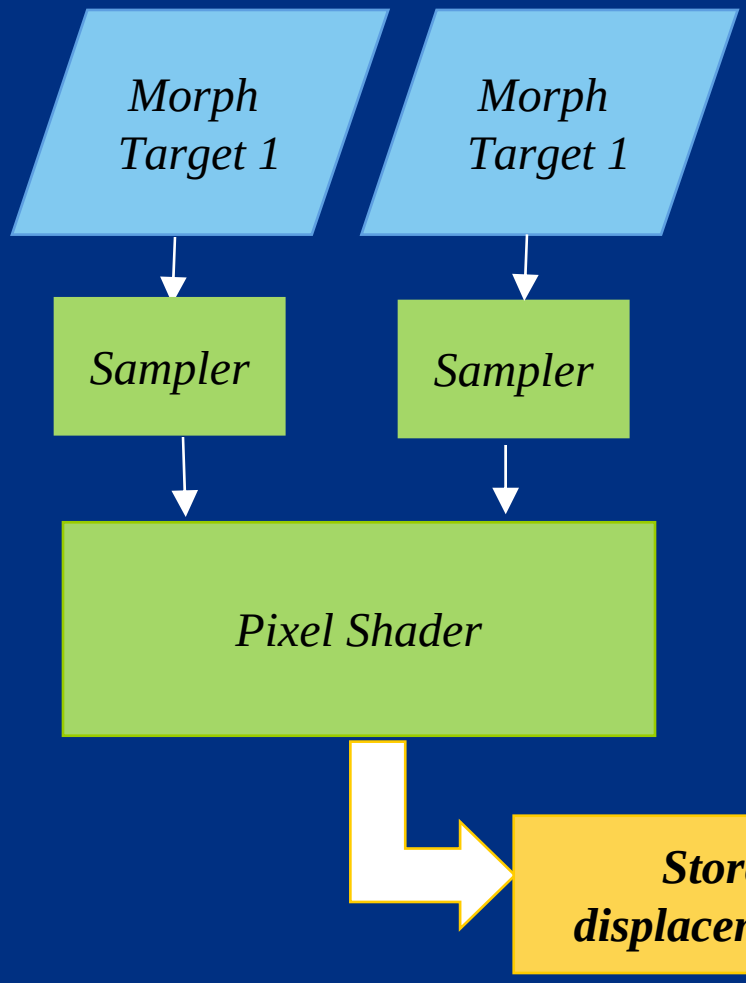
- **Want to animate on control cage: low polygon model**
  - Allows more complex animation computations
  - Higher quality animation with more animation data
  - Can store animated mesh per-frame for later operations
  - Can re-use animated objects from vertex buffer for shadows, reflections, etc.
- **Tessellation post animation**
  - Animation post tessellation doesn't even make sense
  - Generates new vertices and transforms them into screen space
  - Allows more complex pixel shaders
  - Allow higher resolution textures



Control Cage Animation Pre-Pass

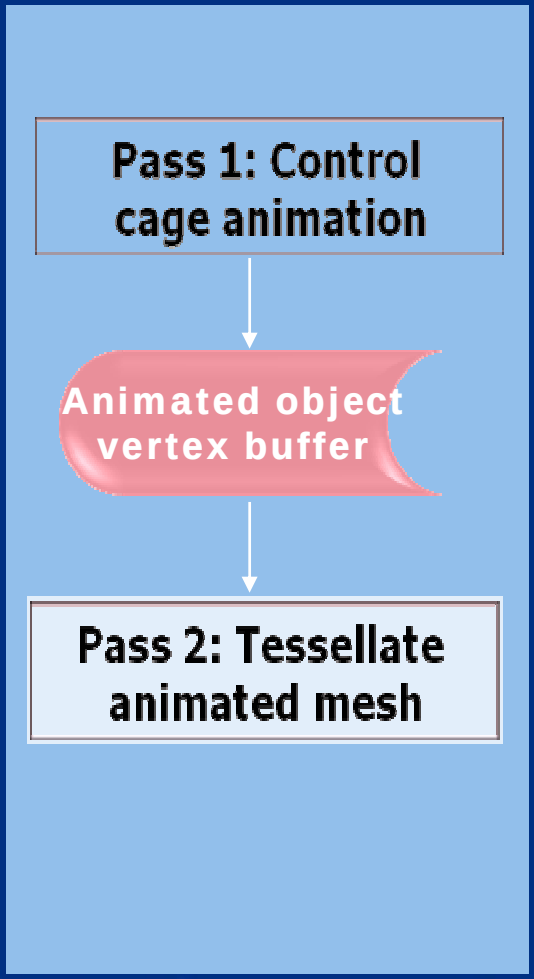
- Can use skinning or morphing
- Apply sparse morph targets technique
  - Different poses of the same mesh
- Morph data is stored in morph textures

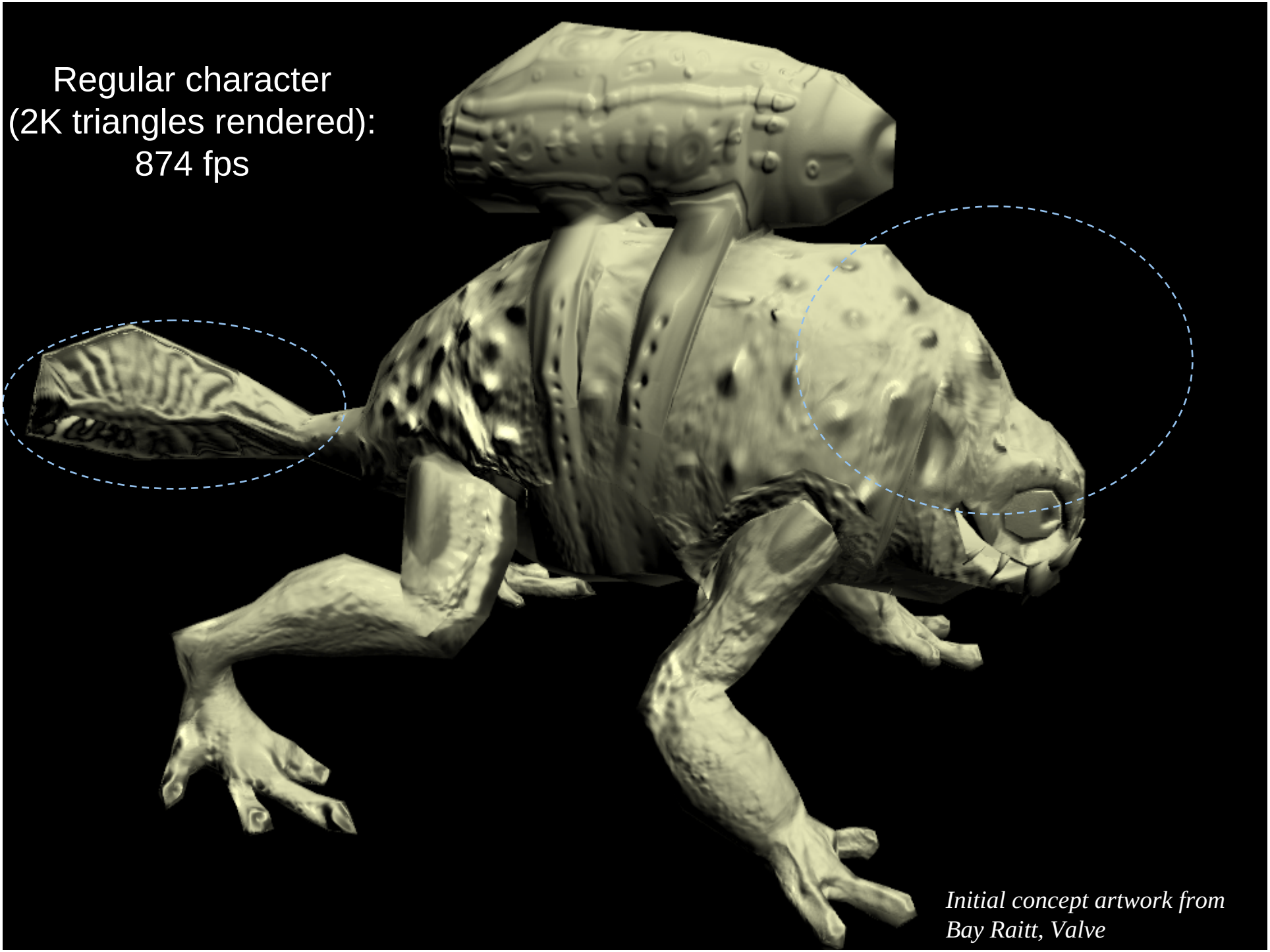
Control Cage Animation: Morphing

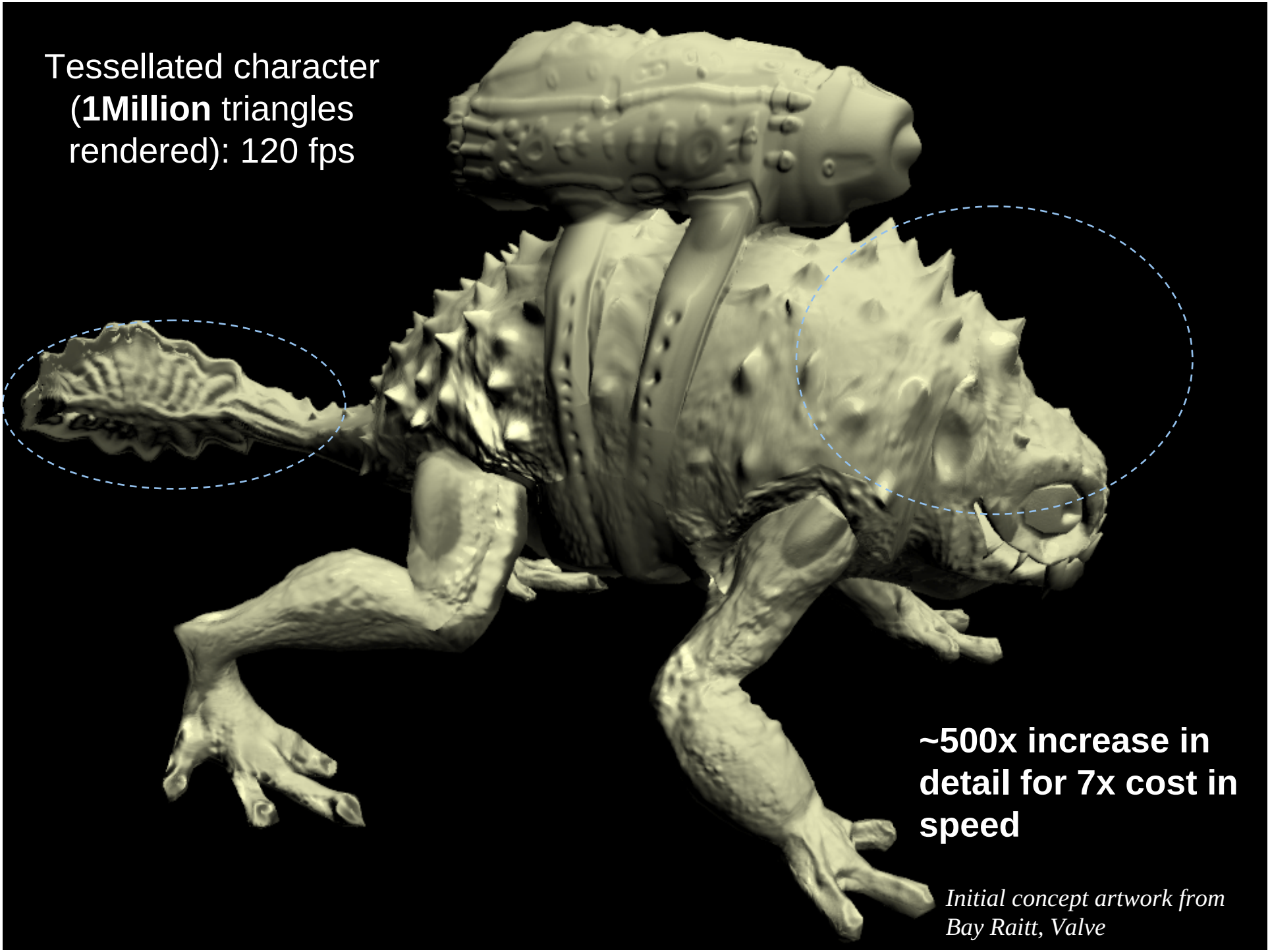


Tessellation After Control Cage Animation

- Goal: Feed animated data into the tessellator
- Option1:
  - Combine vertex positions with morph displacements in a separate pass
  - Cycle the data out
- Option2:
  - Just apply morph displacements directly in the evaluation shader
  - Can be wasteful




Regular character
(2K triangles rendered):
874 fps
Initial concept artwork from
Bay Raitt, Valve


Tessellated character (**1Million** triangles rendered): 120 fps
~500x increase in detail for 7x cost in speed
Initial concept artwork from Bay Raitt, Valve

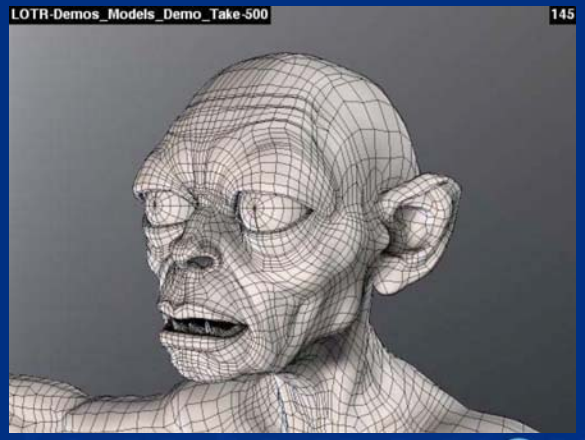
Lighting Directly from Displacement Map

- Use central differences to approximate the derivative field
  - Compute per-pixel normals based on the per-vertex tangent frames based on current displacement
- Memory savings at the cost of additional ALU ops
  - Instead of storing an additional normal map
- Higher quality resulting lighting
- Can support dynamic height fields
  - Great for terrains / destruction
    - See Johan Andersson's talk on terrain rendering

Character Creation Pipeline in Film

- Subdivision surface geometry is the technology for all modeling work
- Allows much more control over model building for animation and texturing
- Create skeletal framework for creature rigging and animation
- Muscles are attached, build up into a complete form, and then layered with skin
- Shading

Gollum, “The Lord of the Rings”, courtesy of New Line



Catmull-Clark Subdivision Surfaces Approximation on Bicubic Patches on GPU

- Need to represent the same subdivision methods as widely adopted by existing tools' pipelines
- New work by Loop and Schaefer from 2007, released as Microsoft Research technical report
- Uses very few Bezier patches to closely approximate Catmull-Clark surface
- For each quad face of the control mesh constructs a geometry patch and a pair of tangent patches which are used to tessellate the surface
  - Handles discontinuity so that we can do high quality displacement
- Uses patch tessellation HW for real-time implementation

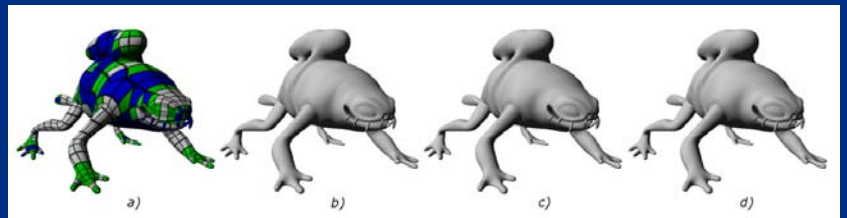


Figure 1: *a)* The patch structure of a Catmull-Clark subdivision surface. The grey patches only contain vertices of valence 4, green have one extraordinary vertex and blue have more than one extraordinary vertex. *b)* Our approximation to the Catmull-Clark subdivision surface using geometry patches and *c)* our final approximation using geometry and tangent patches compared with *d)* the actual Catmull-Clark limit surface.

[Loop Schaefer 2007]

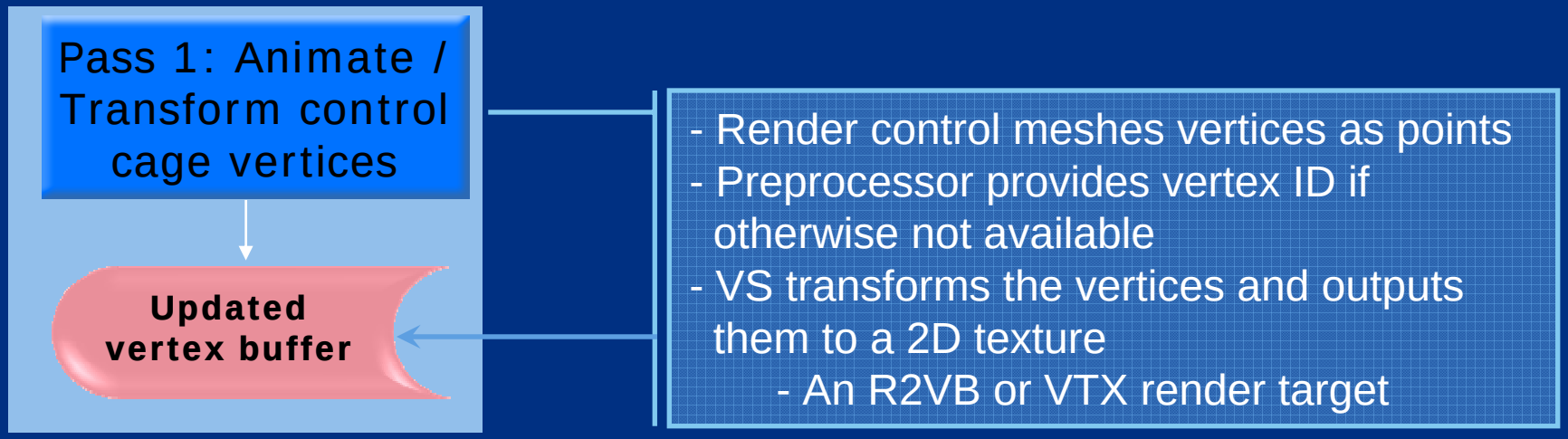
Rendering Tessellated Terrain

- Rendering terrain has a lot of challenges
  - Very large models, continuous spans across space
  - Rendered simultaneously very close and far away
  - Necessitates view-dependent LOD

DICE: Battlefield 2

- Tessellation pipeline gives that control and flexibility
  - Compute tessellation factors in a pre-pass
  - Supply them to the tessellation pass for adaptive tessellation

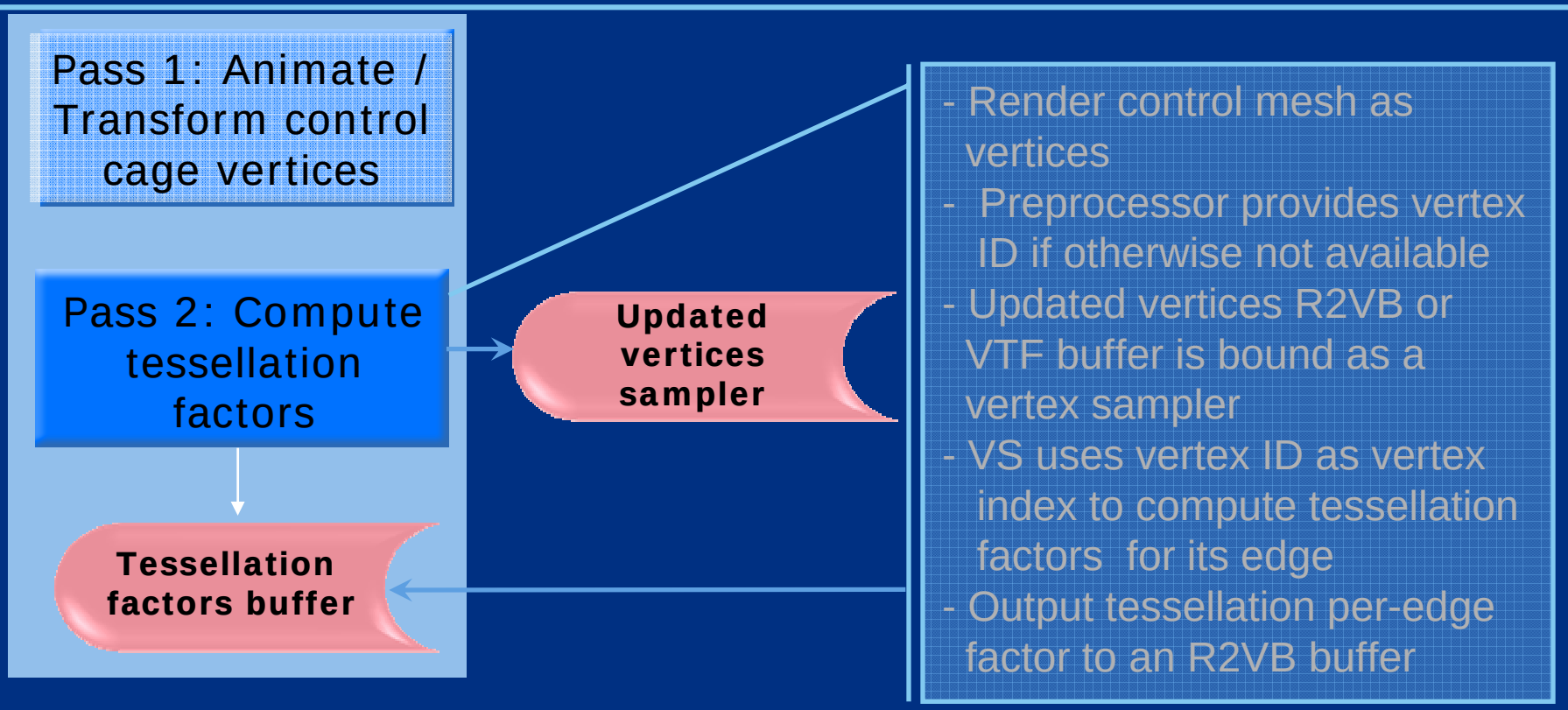
Adaptive Tessellation Rendering



Vertex Prepass Code Sample

```
// Compute displaced position
float3 vPositionOS = DisplaceVertex( i.vPositionOS.xyz, i.vTexCoord, vNormalWS );

// Transform displaced vertex position to camera space:
float4 vPositionSS  = mul( mV,  float4( vPositionOS.xyz, 1 ) );
       vPositionSS /= vPositionSS.w;

o.vPositionSS = vPositionSS.xyz;

// Compute output position for rendering into a texture
float2 vTformVertsMapSize = float2( 1024, 16 );

o.fVertexID       = i.vPositionOS.w;
int nVertexID     = floor( i.vPositionOS.w );
int nTextureWidth = vTformVertsMapSize.x;

float2 vPos   = float2( nVertexID % nTextureWidth, nVertexID / nTextureWidth );  // Compute row and column of the position in 2D texture
       vPos  /= vTformVertsMapSize.xy;
       vPos.y = 1.0 - vPos.y;
       vPos   = vPos * 2 - 1.0;                                                  // Move to [-1; 1] range

o.vPositionCS =  float4( vPos.xy, 0, 1 );
```

Adaptive Tessellation Rendering



Computing Tessellation Factors

```
// Current vertex ID:
int nCurrentVertID = (int)( i.vPositionOS.w );

// Determine the ID of the edge neighbor's vertex (remember that this is done with non-indexed primitives, so
// basically if the current vertex is v0 ,then we have edges: v0->v1, v1->v2, v2->v0

// Manual implementation of MOD works for integer values
int nCurrentVertEdgeID = nCurrentVertID - 3 * floor( (float)nCurrentVertID / (float) 3);

int nEdgeVert0ID = nCurrentVertID;     // this works if current vertex is v0 or v1
int nEdgeVert1ID = nCurrentVertID + 1;

if ( nCurrentVertEdgeID == 0 )
{
    nEdgeVert0ID = nCurrentVertID + 1;
}
else if ( nCurrentVertEdgeID == 1 )
{
    nEdgeVert0ID = nCurrentVertID + 1;
}
else if ( nCurrentVertEdgeID == 2 )                // In case of v2 we need to wrap around to v0
{
   nEdgeVert0ID = nCurrentVertID - 2;
}
```

Compute current edge's end-points' indices

Computing Tessellation Factors

```
// Compute the fetch coordinates to fetch transformed positions for these two vertices to compute their edge statistics:
//

float2 vTformVertsMapSize = float2( nTformedVertsWidth, nTformedVertsHeight );
int    nTextureWidth      = vTformVertsMapSize.x;

// Vertex0: nCurrentVertID, compute row and column of the position in 2D texture:
float2 vVert0Coords   = float2( nCurrentVertID % nTextureWidth, nCurrentVertID / nTextureWidth );
       vVert0Coords  /= vTformVertsMapSize.xy;

// Vertex1: nEdgeVert0ID, compute row and column of the position in 2D texture:
float2 vVert1Coords   = float2( nEdgeVert0ID % nTextureWidth, nEdgeVert0ID / nTextureWidth );
       vVert1Coords  /= vTformVertsMapSize.xy;

// Fetch transformed positions for these IDs:
float4 vVert0Pos = tex2Dlod( sTformVerts, float4( vVert0Coords, 0, 0 ));
float4 vVert1Pos = tex2Dlod( sTformVerts, float4( vVert1Coords, 0, 0 ));

// Swap vertices to make sure that we have the same edge direction regardless of their triangle order (based on vertex ID):
if ( vVert0Pos.w > vVert1Pos.w )
{
   float4 vTmpVert = vVert0Pos;
   vVert0Pos = vVert1Pos;
   vVert1Pos = vTmpVert;
}
```

Fetch the edge's endpoints from transformed vertices texture, then compute tessellation factor with your algorithm and output to PS / render target

Adaptive Tessellation Rendering

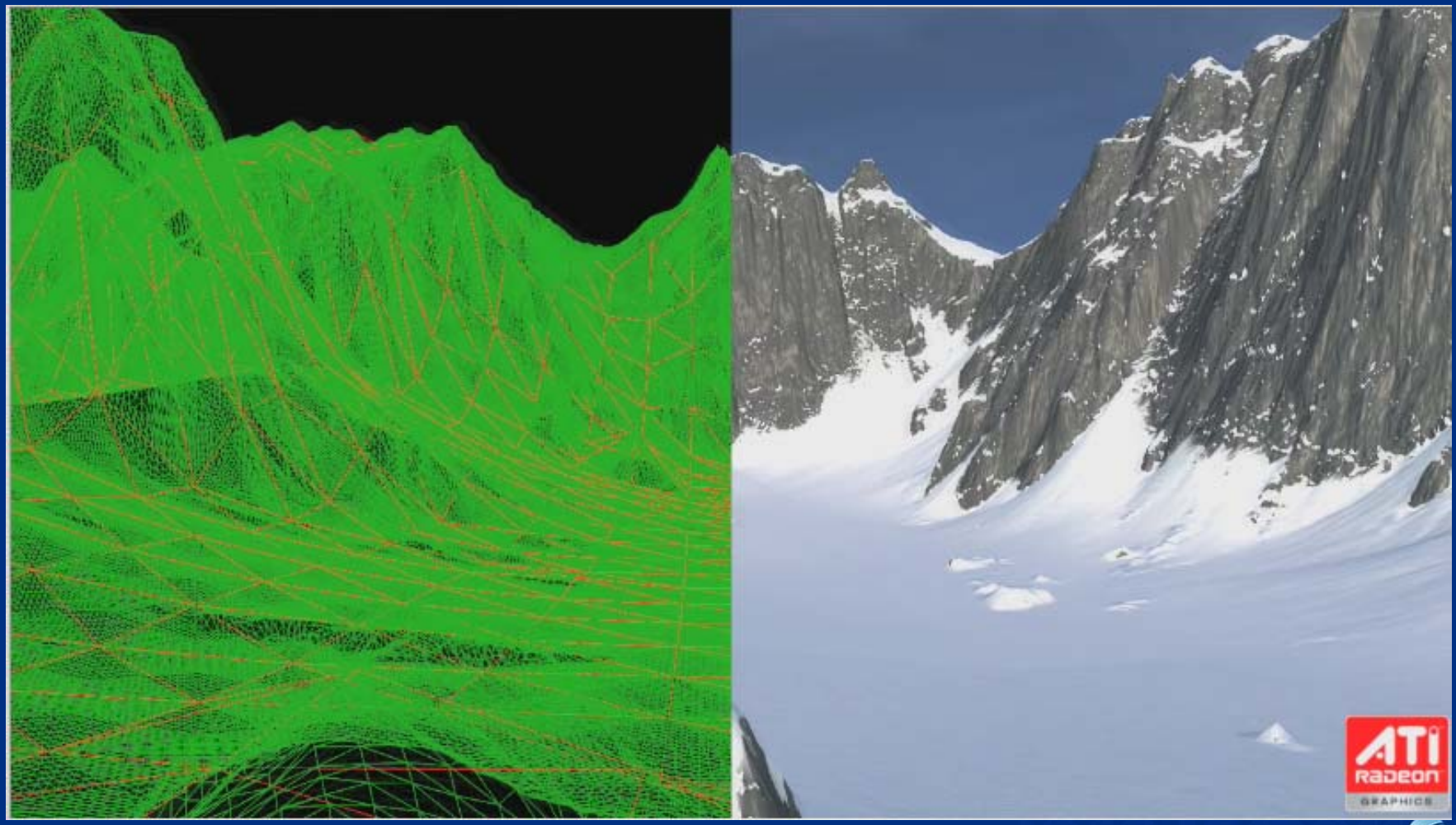
Pass 1: Animate / Transform control cage vertices

Pass 2: Compute tessellation factors

Pass 3: Tessellate and render

- Vertex buffers bound:
  - Input or transformed control cage as one stream
  - Tessellation factors as another stream
- Hardware / driver automatically use tessellation factors to generate domain locations and propagate them to VS

Terrain Tessellation Demo



Terrain Rendering: Comparison

	Low Resolution with Tessellation	High Resolution, No Tessellation
On-disk model polygon count (pre-tessellation)	840 triangles	1,280,038 triangles
Original model rendering cost	1210 fps (0.83 ms)	
Actual rendered model polygon count	1,008,038 triangles	1,280,038 triangles
VRAM Vertex buffer size	70 KB	*31 MB*
VRAM Index buffer size	23 KB	*14 MB*
Rendering time	821.41 fps (1.22 ms)	301 fps (3.32 ms)

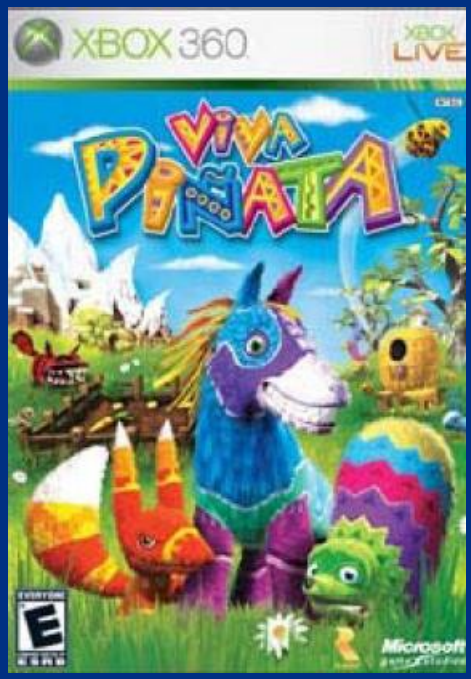
Both use the same displacement map (2K x 2K) and identical pixel shaders

Rendering with tessellation is > 6X faster and provides memory savings over 44MB! Subtracting the cost of shading

Results collected on ATI Radeon HD 2900 XT

Porting from Xbox 360 to PC

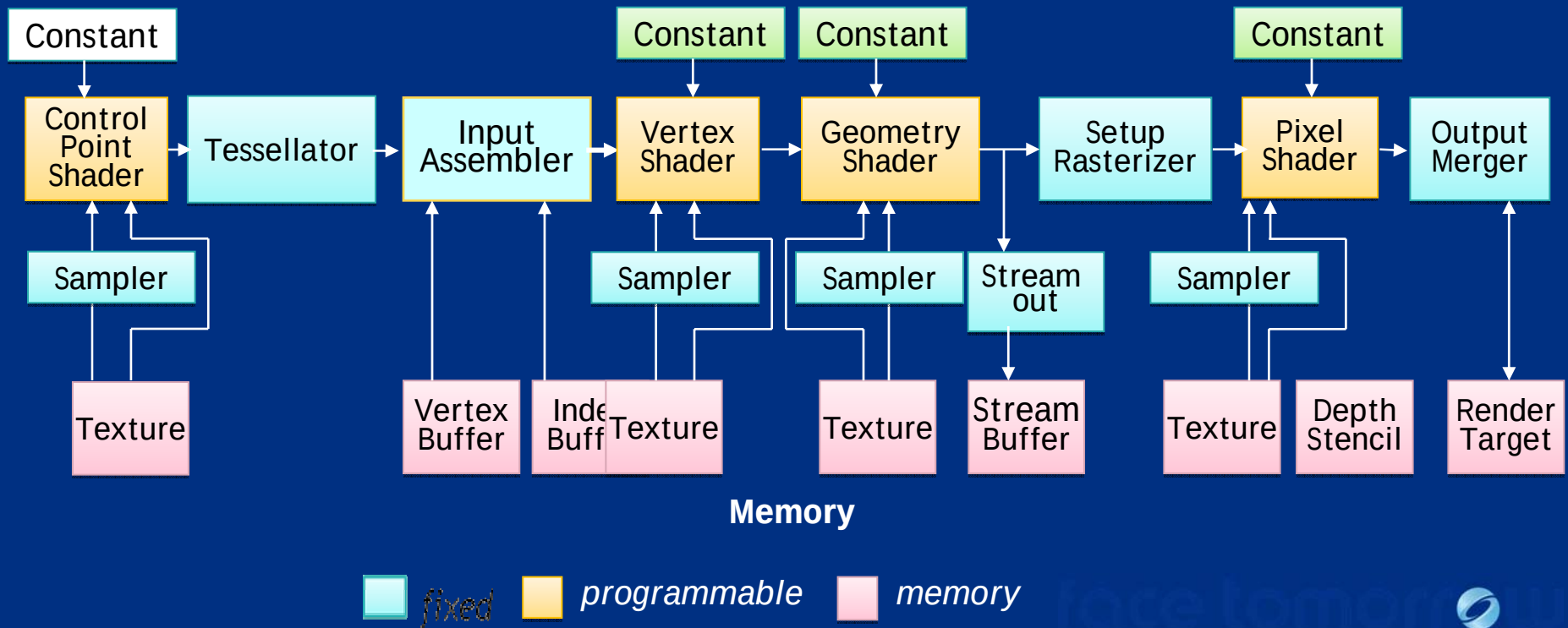
- Xbox 360 and ATI Radeon HD 2000 Series share the same tessellation hardware
- Games which currently use tessellation on Xbox 360 can take advantage of this feature directly
  - Ex: Viva Piñata
  - See Michael Boulton's talk on tessellation in Viva Piñata



Future API Directions Will Expose Tessellation

- Microsoft's future direction includes tessellation support in the upcoming APIs

Logical Pipeline Evolution, Chas Boyd, "The Future of DirectX", GDC 2007 presentation



Cinematic Rendering Relies on Tessellation for Quality and Control

Currently film and games differ in geometry management

- Cinematic rendering relies on extreme details
- Previously, games couldn't afford this luxury

We would like to change that now!

Both must manage details for stable performance

Bring tessellation techniques from film in real time rendering scenarios

- Takes advantage of highly efficient tessellation HW and memory bandwidth of ATI Radeon HD 2000 Series
- Fast displacement mapping and animation

Geri's Game, Pixar

Acknowledgements

- Josh Barczak, Abe Wiley, Dan Roeger from AMD Game Computing Applications Group
- Vineet Goel, Tom Pringle and Dmitry Semiannikov, Raja Koduri, Phil Rogers (AMD)
- Charles Loop and Peter-Pike Sloan (Microsoft)

References

- Catmull, E. and Clark. E. Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. Computer Aided Des. 10, pp. 350-355 (1978)
- Doo, D., and Sabin, M. Behavior of recursive division surfaces near extraordinary points. Computer Aided Design 10, pp. 356-360 (1978)
- Loop, C. Smooth subdivision surfaces based on triangles. Master's Thesis, University of Utah, Department of Mathematics, 1987
- DeRose, T., Kass, M., and Truong, T. Subdivision Surfaces in Character Animation. Proceedings of SIGGRAPH 98, Computer Graphics, Annual Conference Series, pp. 85-94, 1998
- Lee, A., Moreton, H., and Hoppe, H. Displaced Subdivision Surfaces, In Proceedings of SIGGRAPH 2000, pp. 85-94, 2000.
- Zorin, D., Schröder, P., DeRose, T., Kobbelt, L., Levin, A., and Sweldens, W., 2000. Subdivision for modeling and animation. SIGGRAPH'00 Course Notes.

References (cont.)

- Yee, H., Hard, D., Preetham, A. J. Procedural Shaders: A Feature Animation Perspective. Game Developers Conference 2004. presentation, San Jose, CA, March 2004.
- Loop, C. and Schaefer, S. Approximating Catmull-Clark Subdivision Surfaces with Bicubic Patches. *Microsoft Research Technical Report*, MSR-TR-2007-44. 2007
- Boyd, C. The Future of DirectX, Game Developers Conference 2007 presentation, San Francisco, CA, March 2007.
- Boulton, M. Tessellation in Viva Piñata, Siggraph 2007 Course 28: Advanced Real-Time Rendering in 3D Graphics and Games. 2007
- Andersson, J. Terrain Rendering in Frostbite using Procedural Shader Splatting, Siggraph 2007 Course 28: Advanced Real-Time Rendering in 3D Graphics and Games. 2007

Questions?

Thank You!