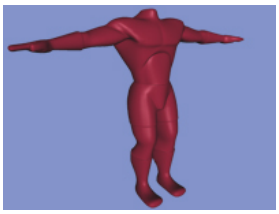
SubD10 Sample

Collapse All



Path

Source	*SDK root* \Samples\C++\Direct3D10\SubD10
Executable	*SDK root* \Samples\C++\Direct3D10\Bin\ *x86 or x64* \SubD10.exe

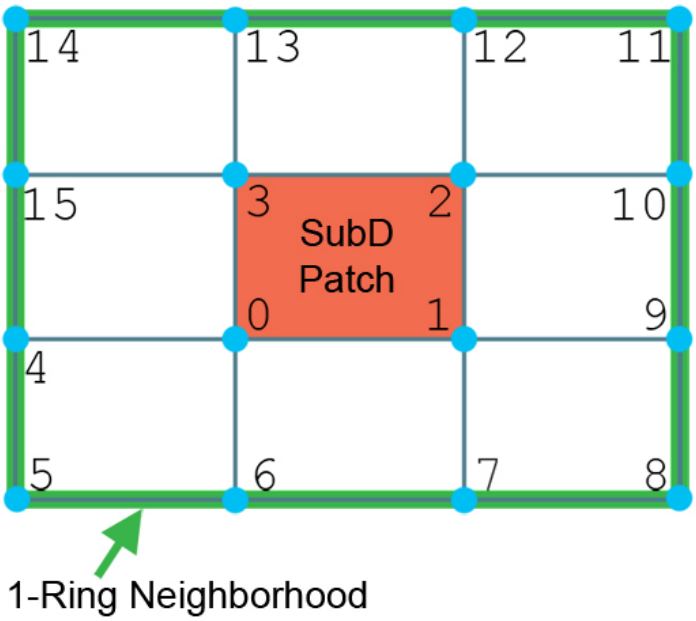
Sample Overview

The SubD10 sample implements the algorithm described in the paper "Approximating Catmull-Clark Subdivision Surfaces with Bicubic Patches" by Charles Loop and Scott Schaefer (available from the link to Microsoft Research's Web site). In this topic, we will refer to this method as *ACC*.

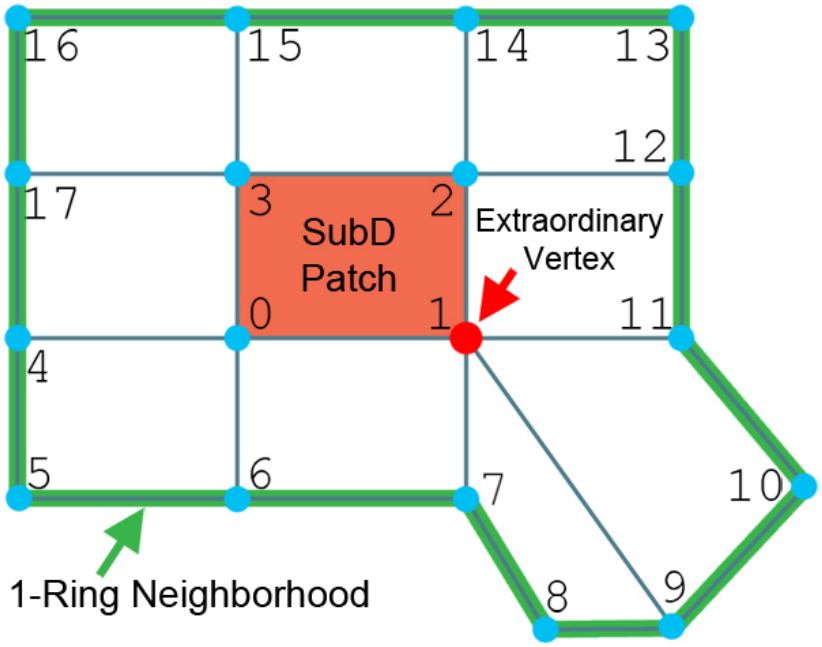
The process is as follows: First, object (.obj) files are loaded into the sample. The object file format was used because it is a common format that can output surfaces as quads. The .obj files are converted into an internal representation. During each frame, this internal representation is skinned and converted to a bicubic patch on the GPU. The bicubic patch control points are streamed out to GPU memory. In the second pass, the bicubic patch control points are read by the GPU and used to construct a close approximation to the Catmull-Clark subdivision surface.

Representing a Catmull-Clark Subdivision Surface

Currently, this sample can only work on Catmull-Clark surfaces that are represented by quads. To convert from the subdivision surface representation to a bicubic patch representation, the ACC method presented in the paper requires the 4 vertices of the subdivision surface quad as well as the 1-ring neighborhood of vertices surrounding that quad. The sample loads object files that contain quads and constructs the 1-ring neighborhood for each quad. Subdivision quads are split into two categories. The first type are regular quads. The 4 vertices of these regular quads all have 4 neighbor vertices (or are valence-4).



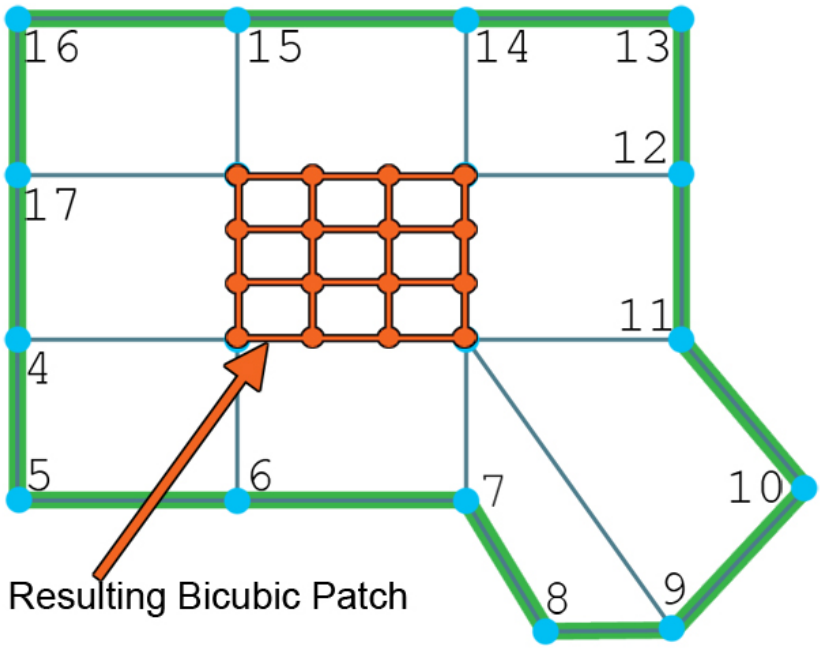
The second type of quads are extraordinary quads. Extraordinary quads have at least one vertex that does not have exactly 4 neighbor vertices. These vertices are called extraordinary as well.



In the diagrams, the vertices are numbered in an inner-to-outer, counter-clockwise spiral. This ordering is used in the conversion from subdivision surface to bicubic patch as a way to keep track of where we are in the conversion process. SubDMesh.h and SubDMesh.cpp contain all of the code for loading object files and converting them into the appropriate representation for ACC.

Converting from a Subdivision Surface to a Bicubic Patch

The mathematics behind the conversion from Catmull-Clark subdivision surfaces to a bicubic representation is too complex for this article. For an in-depth explanation, please see the paper linked above. To summarize the paper, each subdivision surface quad is converted to a bicubic representation. A mesh made up of 100 quads will, therefore, be converted to 100 bicubic patches. Each control point of the bicubic surface is a weighted combination of several of the points from the subdivision surface quad at it's 1-ring neighborhood. The numbering scheme helps facilitate which input points and weights are used to construct each output bicubic patch control point.



This sample converts from the Catmull-Clark representation to a bicubic surface representation using the GPU. All information for one subdivision surface quad is stored in one vertex. This includes the actual quad vertices and the entire 1-ring neighborhood. For regular patches, the actual data is stored in the vertex. This works because regular patches only need to store 16 positions (4 inner vertices, plus 12 outer). Extraordinary patches may require more than 16 positions. Unfortunately, Direct3D 10 does not allow a vertex to have more than 16 elements. For extraordinary patches, we store the indices to the inner 4 quad vertices as well as the outer 1-ring neighborhood. During conversion, the actual values are fetched from a vertex buffer.

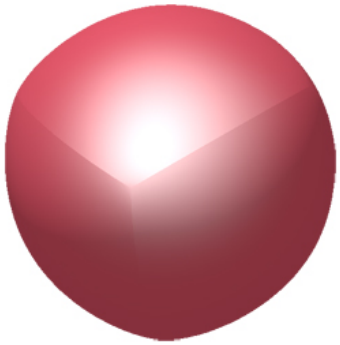
SubDtoBezier.fx contains all of the techniques that convert from the Catmull-Clark subdivision surface to the bicubic patch. While there are several different variations of VSConvertToBezier to show off different methods of conversion, they all follow the same basic formula. First, LoadVerts is called to load the vertices. For extraordinary patches, the inputs to the VS are used to determine which vertices to fetch. For regular patches, the inputs to the VS are used directly. Additionally, the vertices can be skinned inside LoadVerts if Skin Meshes is selected in the UI.

Second, the bicubic patch control points are generated by using functions in GenPatchExtra.fxh and GenPatchRegular.fxh. These two files contain functions that implement parts of the ACC algorithm.

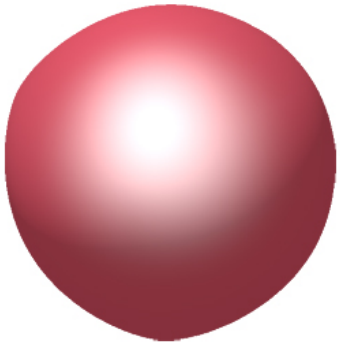
Lastly, all control points for the bicubic patch are written into 1 vertex. That vertex is streamed out to a buffer. Since there will never be more than 16 control points for the bicubic surface, we don't have to worry about writing out more than the number of elements that a vertex can have in Direct3D 10.

Fixing the Normals

For regular patches, the conversion from subdivision to bicubic patch produces a bicubic surface that is the exact representation of the Catmull-Clark limit surface for that patch. However, for extraordinary patches, the bicubic patch produced is an approximation that does have some error. While the resulting mesh will be watertight, the derivatives across patch boundaries will not be continuous. This results in shading artifacts as shown in the following figure.



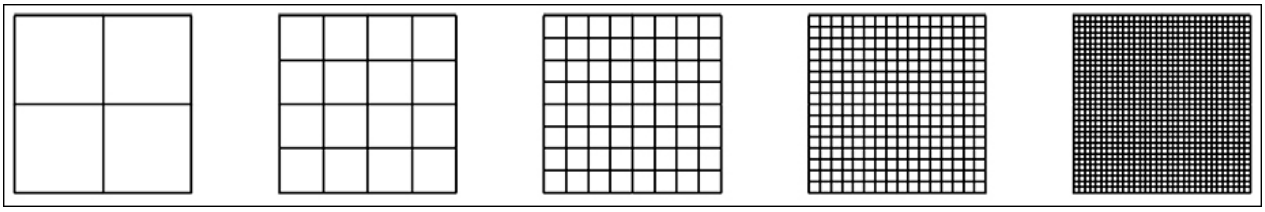
ACC corrects this by also creating tangent and bitangent patches for each bicubic patch. These tangent and bitangent patches (or U and V patches) are used in place of the derivatives of each bicubic patch. While the underlying geometry doesn't change, the tangent and bitangent patches correct the shading and give a very good approximation of the Catmull-Clark surface.



The generation of tangent and bitangent patches requires an extra pass since the data for the bicubic patch is already filling up 16 elements. Fortunately, this generation is only necessary for extraordinary patches. Separating regular and extraordinary patches also allows further optimization, since regular patches have well-known structure and can be optimized accordingly.

Reconstructing the Bicubic Patch

Once the bicubic (and possibly tangent/bitangent) patch control points have been streamed out, all that is left to do is to reconstruct the bicubic surface. This sample uses a series of pre-tessellated squares. The squares contain UV coordinates in the range of [0..1] and also contain some of the precomputed parts of the Bernstein basis functions to help save some computation on the GPU. The sample creates one tessellated square for each level of tessellation that will be required.



For each mesh, which tessellated square to use is determined based on the **Patch Divisions** slider. For example, if **Patch Divisions** is set to 8, then the application selects the square that is tessellated into 8×8 quads. In addition, if **Distance Attenuate** is selected, distance to the camera is used to augment the number of patch divisions to use.

This square is then instanced across the mesh, once for each bicubic surface that was output from the previous passes. In the vertex shader in SubD10.fx, the InstanceID is used to determine which set of bicubic surface control points to fetch. Then, for each vertex in the pre-tessellated square, the bicubic control points are used along with the UV [0..1] stored in the vertex to reconstruct the bicubic surface at that point. For regular patches, the surface normal is computed using the two partial derivatives of the bicubic surface in U and V and performing across product on them. For extraordinary patches, the tangent and bitangent (U and V) patches are used to reconstruct the partial derivatives which are then crossed. This is also where bump mapping and displacement mapping happen if the technique and model support it.

Additionally, the sample supports another way to reconstruct the bicubic surface without using pre-tessellated patches. The USE_PRECALC_PATCHES define at the top of SubD10.cpp can be set to 0 to show this alternative method. The alternative method binds a NULL vertex buffer and calls **DrawIndexedInstanced** (*VerticesToDraw* , *NumPatches* , 0, 0, 0). Although no vertex buffer is bound to the pipeline, the VS gets a series of VertexIDs and InstanceIDs passed into it from the InputAssembler. The InstanceID is again used to determine which set of bicubic patch coefficients to use. The VertexID is used to determine the UV for that point, and the UV is used like before to reconstruct the bicubic patch.

What About the Geometry Shader?

It is also possible to reconstruct the bicubic surface without pre-tessellated squares and without using the Input Assembler. The Geometry Shader could be used to tessellate over the [0..1] UV domain and output multiple triangles. This would do away with the need for pre-tessellated geometry or Input Assembler tricks. There are two problems with this. The first is that the Geometry Shader is limited to only 4096 bytes of output. That means that tessellation levels above 5 divisions per side are not possible with the current sample. The second is performance. The performance of the Geometry Shader in current Direct3D 10 hardware diminishes as the amount output from the Geometry Shader increases. Currently, the instanced squares method is an order of magnitude faster than the comparable Geometry Shader implementation on the same mesh. For this reason, the Geometry Shader implementation was removed from the sample.

Performance Considerations

The conversion from Catmull-Clark to bicubic patches is linear with respect to the number of patches. Because regular patches have well-known properties, we can split them out into their own pass in order to apply optimizations to regular patches that would not apply to extraordinary patches.

The larger cost of the algorithm is in the evaluation of the bicubic surface. The evaluation of the bicubic surface must take place for each point in the tessellated patch. For a regular patch, this evaluation requires 16 loads from a buffer and approximately 73 ALU operations. For an extraordinary patch, the cost can go up to 24 loads from a buffer and approximately 120 ALU operations if the tangent and bitangent patches are stored sparsely.

Because of the bicubic evaluation cost, it is not entirely intuitive which rendering operations could be sped up by using ACC. At first glance, simple 4 bone skinning seems to be an obvious candidate. Skinning the much lower resolution Catmull-Clark representation and then converting that to bicubic patches appears cheaper than skinning a very high-resolution pre-tessellated version of the same mesh. However, the cost of evaluating the bicubic patch outweighs the cost of skinning. Both will happen at the same frequency. Skinning a pre-tessellated mesh will require 16 constant buffer accesses (12 if you ignore scaling) and approximately 38 ALU operations to skin, rotate normals, and so on. Although this happens for each vertex of the pre-tessellated mesh, it is still cheaper than evaluating the bicubic surface. Implementing both options shows roughly a 30% performance increase when skinning the pre-tessellated mesh versus skinning the Catmull-Clark representation and then converting that to bicubic patches.

From a performance standpoint, any operation that would be more costly on a high-resolution tessellated mesh than bicubic evaluation would be a good candidate for ACC. This could include more advanced types of skinning including animation blending, morph targets, or even cloth simulations.

What Do All of the Options Do?

Patch Divisions
: Determines the default number of divisions per-side used to tessellate each bicubic patch.

BumpHeight
: Changes the displacement height for meshes and techniques that support it.

Technique dropdown
: Changes the current technique. See the next section in this topic for a list a description of the techniques.

Toggle Wires
: Enables wireframe overlay.

Separate Reg/Extra
: Whether or not to convert regualr and extraordinary patches in separate passes. Enabling this is generally a good optimization.

Convert Every Frame
: Perform the subdivision to bicubic patch conversion every frame.

Distance Attenuate
: Modifies Patch Divisions per-object based on the distance to the camera. Farther objects need fewer Patch Divisions than nearer objects.

Skin Meshes
: Enables skinning of the subdivision mesh.

What Do All of the Techniques Do?

RenderScene
: Renders the scene using the ACC approach without using tangent and bitangent patches to correct shading.

RenderSceneTan
: Same as RenderSceneTan but the tangent and bitangent patches are expressed in a compact form. This saves bandwidth at the expense of more ALU operations in the bicubic reconstruction.

RenderSceneBump
: Same as RenderScene, but enables displacement and bump mapping on objects that support it.

RenderSceneBumpTan
: Same as RenderSceneTan_Compact, but enables displacement and bump mapping on objects that support it.

What Do All of the #defines Do?

MAX_BUMP
: This is the maximum bump amount for displacement mapping (x1000).

MAX_DIVS
: The maximum number of patch divisions per-side.

NUM_BONES
: Objects are rigged programmatically. This determines the number of bones to use per-object.

MAX_POINTS
: Maximum number of points that can be references by a subdivision patch. This includes the 4 quad points as well as the entire 1-ring neighborhood.

NUM_REGULAR_POINTS
: Number of points that are referenced by a regular patch. This is not variable.

MAX_VALENCE
: This is the maximum vertex valence (number of neighbors) that we expect to see from any vertex in the subdivision surface.

Artwork

The monsterfrog and bigguy models used in this sample are courtesy of Bay Raitt, Valve Software.