B I J A N   F O R U T A N P O U R

alternative geometry

THINKING OUTSIDE THE TRIANGLE

Thanks to science fiction-like advances in technology, games have taken a large step forward in image quality and photorealism during the past ten years. Games are the killer app of graphics hardware, and are a huge part of the raison d'etre of that industry. In turn, if video cards did nothing more than drive the monitor, we'd all still be playing PONG. Because graphics hardware development and game development are so intertwined, many of the decisions made by one side affects the other. A primary example is the use of the triangle as the core rendering primitive. GPUs have been designed to process hundreds of millions of triangles per second, and so game engines are forced to express the game as triangles, the art tools have to output artistic vision as triangles, and finally, artists have to think in triangles.

Even though rasterization of triangles clearly has a stronghold on today's rendering pipelines, there are other methods and technologies that are available today. They may not be appropriate for rendering the entire game, but they may be appropriate for rendering parts of a 3D scene at interactive speeds. Alternatively, they may be useful in other parts of the game development process, such as tools for creating art assets. In this article, we will review technologies that are alternative to the triangle, but still make sense for use today.

EXPLICIT GEOMETRY

>> First, let's begin with a brief summary of the drawbacks of using triangles. Triangles are very explicit; a lot of data is required to describe a single triangle. The location, normal, and texture coordinate of each vertex add up to a lot of data very quickly, especially for models with curved surfaces. This produces a bottleneck for data transfer between CPU and GPU, not to mention a lack of storage space in video memory for all that data.

This leads to the need for a series of programming optimizations to help manage the data efficiently. Examples include triangle stripping, creating data-aligned vertex buffers for position, normal, and texture coordinates, followed by index buffers to access them correctly, and sorting vertices such that they are used in sequential order to minimize missed hits to the hardware vertex cache. It's not really what one would consider part of the pleasures of graphics programming, unlike creative lighting, materials, and effects algorithms.

Sorted triangles with per-vertex and connectivity information are at the lower levels of the language of expressing geometry. One could think of it as akin to programming in assembly. You can express anything in assembly, and if used correctly it is faster than any other language, but it is way too detailed and explicit a language to be fun or practical for daily coding of large systems. Yes, given current hardware architectures, any form of geometry supplied to the system will eventually be converted to triangles before final display anyway,

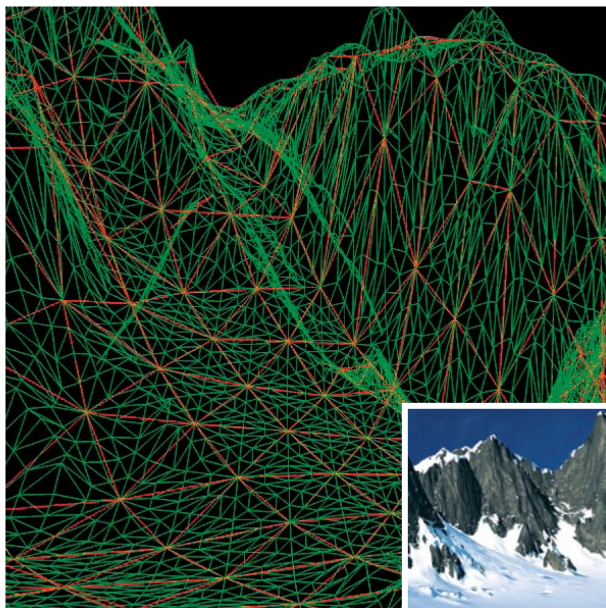


FIGURE 1 An example of real-time tessellation on a GPU from AMD's ATI Radeon HD 2000 series demo.

but that shouldn't necessarily dictate the need to push triangles all the way up to the engine and art pipeline.

PARAMETERIZED GEOMETRY

At the other end of the geometry definition spectrum comes the class of parameterized geometry, described by mathematical equations. Curves and curved surfaces fit into this category, as do subdivision surfaces, Bezier patches, NURBS (Non Uniform Rational B-Splines), and quadric surfaces. There are at least four advantages to using parameterized curves and surfaces over triangle meshes. First, they require much less data than explicit meshes do. That means less memory storage, fewer transformation calculations, and less bus bandwidth. Second, they are scalable, providing any amount of detail desired. For instance, a curved patch can be tessellated into just a few triangles, or many thousands of triangles, depending on the distance to camera. The third advantage is that they simply look better, and represent rounded characters, terrain, or terrain objects such as trees or rocks better than triangles do. Lastly, for per-frame calculations such as animation or collision, they require fewer calculations. The geometry hull which contains fewer vertices can be used for any calculations, and then the smoother and higher resolution mesh can be generated and rendered afterward directly by the graphics hardware.

Even though today's GPUs process triangles faster than curved primitives, the advantages of surfaces are compelling enough to encourage researchers and hardware companies to keep trying. One relatively recent idea is N-patches, short for "normal patches," also called PN triangles. The goal of PN triangles is to improve the shading and silhouettes of flat triangles by generating a curved surface to replace each triangle. The curved surface is created by generating new triangles which approximate the curved surface on the fly. This actually can be accelerated by hardware because it can generate the surface using only the normals and positions of the original underlying triangle. No information from neighboring triangles is needed.

ATI in fact has implemented hardware acceleration of N-patches in some of its chips. Beginning with the 8000 series of chips, this hardware feature was called TRUFORM, and was available in many games—QUAKE, QUAKE 2, UNREAL TOURNAMENT 2003 and 2004, MADDEN NFL 2004 and WOLFENSTEIN: ENEMY TERRITORY, among others.

But beginning with the Radeon X1000, the feature was no longer advertised, probably because of a lack of wider acceptance among developers. It also didn't help that NVidia didn't have a similar feature in its hardware at the time. NVidia had a competing tessellation solution based on quintic-RT patches, which did not receive much attention from developers. QuinticRT patches are based on quintic equations—polynomial equations of degree five. Quintic equations have some interesting properties, although I can't help but be reminded of the scene in *Spinal Tap* where Nigel declares, "This one goes to 11," while describing his amp. Usually third and fourth degree polynomials are more than sufficient to represent the needed curvature.

Beginning with the R600 GPU found in the ATI Radeon HD2000–4000 series, new hardware tessellation units were added. The new hardware is placed before the vertex shader stage in the GPU pipeline. It does not calculate the final vertex positions, nor store them in video memory. Instead the tessellator generates new vertices and their positions as parametric coordinates, which are then passed to the vertex shader, along with the vertex indices of the base mesh. It is then the vertex shader's responsibility to convert the parametric coordinates to world coordinates, as well as to apply the desired surface evaluation function. This may be any algorithm for evaluating any higher order surface, such as Bezier, NURBS, or Catmull-Clark subdivision surfaces.

Its important to note that the tessellation is done in one pass through the GPU pipeline, and no extra video memory is needed to store the generated data. The tessellation unit can generate up to 411 triangles from a single source triangle. In one of the technical demonstrations from ATI, realtime tessellation of terrain geometry is performed with a model of 4050 triangles being tessellated to over 1.6 million triangles, with a frame rate of 110 frames per second. See Figure 1 or visit www.gametrailers.com/video/tessellation-tech-ati-radeon/20445 for a video of the demo.

Even though today's GPUs process triangles faster than curved primitives, the advantages of surfaces are compelling enough to encourage researchers and hardware companies to keep trying.

Access to the new hardware is not part of OpenGL or DirectX 10, but is available as an extension to DirectX 9 and Direct X 10 from ATI's website, along with an excellent paper by Tatarchuk, Barczak, and Bilodeau, "Programming for Real-Time Tessellation on GPU" which also discusses the upcoming DirectX 11 tessellation architecture (see http://developer.amd.com/gpu/radeon/Tessellation).

Clearly, there is a vision and desire for graphics hardware handling curved surfaces at interactive frame rates. They haven't become ubiquitous yet, but it may simply be a matter of time. Working with curved patches in a production pipeline does have its drawbacks though. One problem to overcome is the need to maintain continuity of the curved surface between patches. Manipulating the control hull (and thus the surface) of one patch may create tears and seams with neighboring patches. Subdivision surfaces address this problem nicely, as the control hull mesh can have any arbitrary topology and is smoothly evaluated between neighboring faces. Subdivision surfaces have already made

their way into current art pipelines with mainstream art tools such as ZBrush and Mudbox.

Even though there currently doesn't seem to be any effort focused on creating subdivision-based game engines, it clearly is possible. One option is hardware tessellation, mentioned above, but the other is a method known as valente subdivision, by David Brickhill. The technique is a high-speed rendering algorithm for subdivision surfaces that was implemented in a game engine running on a PlayStation 2. The algorithm achieves interactive speeds by avoiding recursion and its disadvantages, namely very inefficiently accessing memory across areas too large to fit into a scratch pad or data cache.

IMPLICIT, VOLUMENTRIC GEOMETRY

>> One of the more interesting methodologies for 3D modeling is voxels. The term voxel is short for volume element, akin to the pixel's meaning of picture element. It follows then that three dimensional space can be thought of as a three dimensional grid of voxels, in the same way that a digital image is a two dimensional grid of pixels. Looking at past usage of voxels in games, aside from the basic definition just mentioned, different people speak of voxels in different terms and contexts. For instance, the assumptions on a voxel set's uniformity of size, adjacency, or alignment requirements are all left to the individual algorithm and context. Even the data contained in a voxel and how it is rendered is open to interpretation.

As an example, NovaLogic Inc, creator of the COMANCHE and DELTA FORCE series, was awarded a patent in 2000 for a voxel-based terrain rendering algorithm used in the VoxelSpace game engine. However, the terrain was actually a 2D heightfield array, and each grid in a two dimensional horizontal grid is extruded to a given height, yielding a tall volume, thought of as a voxel. This algorithm for terrain generation and rendering was very innovative for its time. It used the CPU for the bulk of the work, and produced results that were impossible to achieve with the underpowered GPUs of its time.

Fast forward to present day. CRYSIS' art toolset, Sandbox, includes a terrain creation and editing system, which includes voxel editing capabilities. The particular use of voxels in this context is to allow for freedom from a fixed single-layer topology, and allows for creation of caves, overhangs, and other complicated configurations. The definition of voxels in this context is more in line with the scientific community's definition of a voxel.

NovaLogic's COMANCHE 4 employs voxel-based terrain rendering.

Another great example of the use of voxels in current games is Maxis' SPORE. The creature building part of the game involves constructing an arbitrarily-shaped body, followed by attaching any number of limbs, of varying types. The limbs can be placed anywhere along the body, using any topology the user wishes. This is done using metaballs, also known as blobbies, which are part of a voxel-based algorithm. The difference between metaballs and regular voxels is that instead of each voxel containing only one of two values (inside or outside the model), the voxel defines a weighting factor, or probability that a surface exists. This allows for the creation of a smooth surface attaching and blending two different models together.

True voxels began with the medical visualization community and are used in Magnetic Resonance Imaging to uniformly sample 3D space. Therefore all voxel are adjacent, same-sized cubes, defined by a single number specifying whether they lie inside the model volume or outside.

Voxels have several advantages over polygonal geometry. The first is smaller data size. A voxel is a single element in space, a single number determining if the voxel is inside or outside of the model's volume. Unfortunately, depending on the application, a lot of voxels may be

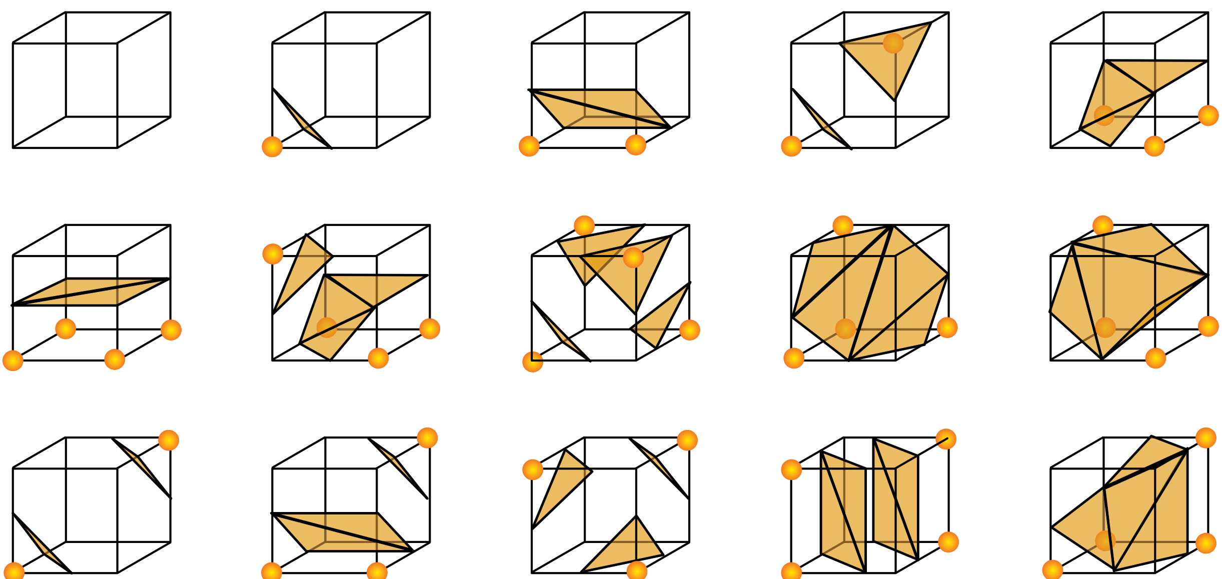


FIGURE 2 The "Marching Cubes" algorithim creates a triangulated isosurface from voxel data.

FIGURE 3 SPORE CREATURE CREATOR uses metaballs, a variation of voxels.

required to render an object or scene, so smart solutions for handling this problem are still required.

The second is its uniform sampling of space, as compared to a polygon. In a polygon-based artist tool for sculpting 3D models, if a vertex is placed too far relative to its neighboring vertices, stretching will occur, which may result in texturing artifacts, and reduced geometry resolution for placing more details, and a possible loss of smoothed curvature. In a voxel-based modeling program, there are no such limitations, as the volume of space is evenly sampled, and new vertices are automatically added to the model surface when the voxels are used to generate the polygonal mesh for rendering.

The third advantage voxels have is that they don't have any topological constraints relative to other parts of the model geometry. Again, this can be very useful when sculpting a character model where limbs and appendages can freely be removed or attached.

There are very few commercial voxel-based 3D sculpting applications available today. One that comes to mind is 3DCoat, which clearly demonstrates the advantages mentioned above.

How to render true voxel data, and whether it can be done at interactive speeds for games, has been the million dollar question. There is also the question of the interaction of voxel data with the remaining elements of a game—for instance, physics calculations of rigid body dynamics, interaction with particles, game characters, and so forth.

One of the problems with voxels is that even though a single voxel may itself may be lightweight, when converted to triangles they can create an incredible amount of data—triangle bloat.

There are a few approaches to rendering voxel data. The first is to create an isosurface, essentially connecting the dots and creating a triangle mesh to be rendered by the GPU. The most well-known method is the "marching cubes" algorithm, which until recently was under patent and not available for wide, unlimited use. However, the patent has expired, and there has been renewed interest by developers. The algorithm basically visits each voxel cube (thus the name "marching cubes"), and generates the needed triangles. It does so by looking at the eight corners of the voxel, which is a single point in space with a number associated with it specifying if it is inside the 3D model (one) or outside (zero). For each of the voxels, if an edge has one vertex with a value of zero and one vertex with a value of one, then a triangle vertex is created at the midpoint of the voxel edge. Adjacent triangle vertices are connected to create a triangle. The triangle creation step is sped up by creating an array of all 256 possible triangle configurations, and indexing into the array. Figure 2 shows the 15 unique cases of the 256 configurations, the rest are rotated or mirrored variations. For instance a cube with vertices valued 0, 1, 0, 1, 1, 0, 0, 0 will produce 01011000 in binary format, or 56 in decimal.

One of the problems with voxels is that even though a single voxel may itself may be lightweight, when converted to triangles they can create an incredible amount of data—triangle bloat. Each voxel can produce up to four triangles, which means twelve vertices. The number of voxels needed to cleanly represent a 3D scene can be quite large, quickly leading to gigabytes' worth of triangle vertex data.

Marching cubes is not the only method for creating triangulated isosurfaces from voxel data. Another algorithm, found in *Graphics Gems III*, "Compact Isocontours from Sampled Data," was used in SPORE's creature builder. Other algorithms exist as well, using other shapes, such as "marching tetrahedrons," originally intended to circumvent the patent on marching cubes.

Much research has gone into fast rendering of voxel data sets on current graphics hardware. The research and implementation challenge comes down to creating a real-time implementation of marching cubes, and there has been significant work done in this area. Interestingly, NVidia released a geometry shader in its GeForce 8800 series of cards, and created a very compelling demonstration in the Cascades demo, discussed at GDC 2007. The demo consists of generating voxel data to create a rock formation by rendering voxel density values into a 3D texture. Then, using the geometry shader, the marching cubes algorithm is used to generate a triangle mesh. Afterward, water particles are dynamically generated and physics calculations are used to determine collision between particles and the rock geometry. Additionally, swarms of dragonflies fly around, avoiding the rock as well. Although a very compelling demonstration of the new geometry shader hardware units, it remains to be seen how effectively it will be used in a 3D game. As a first step, the hardware shaders may be better suited for use in an art tool as part of the source art creation pipeline.

Another interesting project in voxel rendering is "High Speed Marching Cubes Using Histogram Pyramids" by Dyken and Ziegler, and can be found at www.mpi-inf.mpg.de/~gziegler. The HistoPyramid algorithm, which is usually used in data compaction for GPUs, was used here for data expansion of voxels into triangles. The algorithm does not use

a geometry shader, but runs three to four times faster than running marching cubes with hardware acceleration on the geometry shader.

Finally, as mentioned earlier, creation of a triangle mesh isn't the only method of visualizing voxels. The other option is volume raycasting. Like raytracing, rays are cast from the camera, through each pixel on the screen, into the voxel data, sampling it and making the final calculations. The advantage of ray casting is that it inherently removes hidden surfaces as well. Voxels that are not seen are not used, and no computation time is wasted on them. Another benefit is that partially transparent surfaces can be taken into account when calculating the final pixel color. One disadvantage of ray casting is that it does not produce any geometry, which may be desirable for physics or collision calculations.

SPARSE VOXEL OCTREES

» During Siggraph 2008, id Software demonstrated the concepts behind the company's latest game engine, id Tech 6. The goal of the new engine is to achieve an unlimited amount of unique texturing and unique geometry. This is accomplished by ray casting into an octree structure of voxels. Octrees are spatial data structures used to partition three-dimensional space by subdividing a scene into eight octants of identical size, and recursively subdividing only each octant that contains geometry to the smallest desired level of detail. The sparse voxel octree algorithm is conceptually easy to understand, but nontrivial to implement. As part of the art processing pipeline, polygonal source art must first be broken down into voxels, called voxelization. There are different methods of voxelization, such as 3D scan conversion, volume projection, and subdivision.

The created voxels are then hierarchically stored in an octree. Each voxel contains the color of the surface model at that position in space. The octree is now considered to be the object that is rendered, there are no polygons so to speak. During runtime, rays are cast from the camera position through each pixel location on screen, and the octree is sampled. If the voxel in the octree that intersects the ray is larger than the size of a pixel, the subvoxels stored in the lower level of the octree are retrieved and the ray stepping is continued. When the ray stepping intersects a voxel that is less than the size of a pixel, its color is used to set the value of the pixel on screen.

One way to think of sparse voxel octrees is as mip-mapping for geometry. As mentioned earlier, the details of implementation are non-trivial. The final result of creating octree data for a typical 3D scene for a game will be very large, on the order of tens of gigabytes. This is way beyond the memory capacity of graphics hardware of many PC systems. So, the next required system is a streaming paging system designed and optimized for hierarchical storage and retrieval of octree data.

Sparse voxel octrees are intended mostly for creation of large static objects with unique detail, namely terrain. For dynamic, animated objects, more work is required and may not be worth the effort. The animation data would be used to deform the octree voxels, and as rays are cast through the screen pixel into the deformed octree, rays would have to be bent to follow the same deformations. Although possible in theory, in practice it still makes sense to follow the traditional triangle rasterization approach.

Further interesting research similar to sparse voxel octrees has been done in a paper called "Interactive GigaVoxels" by Crassin, Neyret, and Lefebvre and can be found at http://artis.imag.fr/Publications/2008/CNL08. The work is based on a dynamic generalized octree with mip-mapped 3D textures at the lowest level leaves of the octrees.

CONCLUSION

» In reviewing NURBS, subdivision surfaces, and voxels, it's clear that they are viable alternatives to triangles. Although, like triangles, they have their weaknesses, and each method provides a good solution to a specific problem.

FIGURE 4 An example of 3D modelling that uses sparse voxel octrees from Jon Olick's SIGGRAPH 2008 presentation.

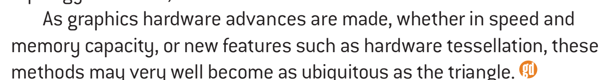
NURBS are ubiquitous in the CAD industry, and are still the preferred method for hard surface modeling of objects with smooth curves. For a car racing game, using NURBS at least in the art pipeline makes sense, both for their ease of use and flexibility when it comes to final tessellation.

Subdivision surfaces avoid the seaming problems of NURBS surfaces, and are a great approach to modeling organic surfaces such as characters and creatures. Subdivision surfaces have been used in at least one game engine on the PlayStation 2, and already have a strong foothold in the art creation pipeline in tools such as Zbrush and Mudbox.

Voxels have traditionally been used for terrain rendering in game engines, as a fast method to avoid using the GPU. Currently voxels are still ideal for authoring terrain because they allow more complex terrain type creation such as caves, arches, and overhangs. They also are ideal for modeling sheer vertical cliffs, since voxels provide uniform sampling of 3D space and avoid the stretched triangle problem.

Voxels can also be used as the basis for a game engine, as they can provide a stylized look to the game and are perfect for implementing completely destructible environments. Additionally, voxels can be used in limited scale in game engines where flexible character geometry topology is needed, as seen in SPORE CREATURE CREATOR.

As graphics hardware advances are made, whether in speed and memory capacity, or new features such as hardware tessellation, these methods may very well become as ubiquitous as the triangle.



BIJAN FORUTANPOUR *is a senior graphics programmer at Sony Online Entertainment in San Diego with 16 years experience in the visual effects and games industries. He is also the author of Enzo 3D paint for Photoshop (www.enzo3d.com). Email him at* **bforutanpour@gdmag.com**.