Abstract. Three-dimensional (3D) models of space objects are commonly described as a mesh consisting of vertices connected to one another. The vertex connectivity is often chosen to create triangular or quadrilateral faces, thus forming a polyhedral representation of the body’s surface. The choice between triangular and quadrilateral faces carries many consequences, as each has its own advantages and disadvantages. Further, certain software pipelines natively work in one format or the other. This work reviews many of the key considerations when choosing the correct 3D model format for solving problems in space science and exploration.

Introduction. Modern space science and space exploration tasks often require three-dimensional (3D) representations of observed celestial bodies. Given the pervasive need for 3D models in common terrestrial applications (e.g., home entertainment, medicine, industrial robotics, self-driving cars, social media), there is a rich literature exploring the many ways to represent 3D surfaces. Most techniques may be categorized as either explicit or implicit methods for representing a surface. This work focuses exclusively on the explicit representation of a surface with a polygon mesh. Although not discussed here, other common explicit representations include non-uniform rational B splines (NURBS) and Bézier surfaces.

A polygon mesh represents a surface as a polyhedron. This polyhedral surface is made up of convex polygon faces—usually a triangle or quadrilateral. The triangular (Tri) mesh and quadrilateral (Quad) mesh conventions are widespread with different pipelines and applications supporting different conventions. A comparison of the same body modeled as a tri and as a quad is shown in Fig. 1. In this work we explore the advantages and disadvantages of each convention, and discuss key algorithms for working with each format.

Remarks on Space Applications. One of the first steps in exploring a new body is to create a 3D model of its shape. Such models already exist for many of the major bodies in our solar system. When we choose to explore new bodies (e.g., comet, asteroid), it is standard practice to construct a low-resolution shape model via lightcurve inversion or radar data before sending a spacecraft. After the exploration spacecraft is launched and as it approaches the body, improved 3D models are generally constructed using optical data and laser ranging data. Oftentimes missions to unexplored bodies move through a sequence of reconnaissance phases, building progressively more detailed models as they slowly step closer towards the body. Such reconnaissance phases may be unnecessary if detailed models exist from earlier exploration missions. The resulting 3D models may be used to address a range of important scientific questions, to choose interesting landing sites (when applicable), and as an input to terrain relative navigation (TRN) algorithms. What follow are a few remarks on especially interesting (to the authors) use cases of 3D models in the space domain.

The customary means of constructing high-quality 3D models of space objects from in-situ image data is stereophotogrammetry (SPG) and stereophotoclinometry (SPC). Both SPG and SPC have been used to model a wide variety of bodies—including the moon and many different small bodies (e.g., Eros, Vesta, and Ceres). Regardless of the method of choice, the resulting 3D model is generally

Figure 1. Three-dimensional (3D) model of Eros using a quadrilateral mesh (top) and triangular mesh (bottom). This 3D model of Eros was developed from NEAR MIS images by Bob Gaskell using the stereophotoclinometry (SPC) pipeline. The PDS-archived 3D model is natively represented as a Quad (in ICQ format), which the authors converted into a Tri for this example.
Mesh Description. A polygon mesh is one of many explicit representations of a continuous surface. A mesh \( M \) consists of a set of vertices \( (V) \), edges \( (E) \), and faces \( (F) \) such that \( M = \{ V, E, F \} \). These sets are defined as \( V = \{ v_i \}_{i=1}^{N_v} \), \( E = \{ e_{ij} \}_{i=1}^{N_e} \), and \( F = \{ f_i \}_{i=1}^{N_f} \). Vertices are connected by edges, where vertex \( v_i \) and \( v_j \) are connected by edge \( e_{ij} \). A collection of edges creates a polygon face, \( f \), where three edges (or vertices) create a triangular face and four create a quadrilateral face. This can be seen in Fig. 3. A collection of faces subsequently creates a polygon mesh, either triangular or quadrilateral (or both) depending on the face shape.

Conventions for Mesh Connectivity. This work focuses on meshes where vertices are connected to one another to form either triangular or quadrilateral faces. As an explicit surface representation, meshes explicitly store the 3D coordinates of each vertex in a data file (referred to here as a vertex table). We also require knowledge of which vertices are connected to each other to form each face (or facet) of the mesh. Such vertex connectivity may be stored explicitly (e.g., in a facet table) or implicitly (e.g., using a specific organization of the vertices). Numerous popular conventions exist for both, and a few of the more important conventions (for the space scientist or engineer) are now discussed.

Explicit Connectivity. The mesh connectivity may be described explicitly in a facet table that records the vertices belonging to each face (three vertices for a tri face; four vertices for a quad face). In general, this approach has the advantage that vertices may be listed in any order stored as a mesh. In the case of models generated using one of the Gaskell-derived SPC pipelines, these meshes are stored as a quad mesh—specifically, using the implicitly connected quadrilateral (ICQ) format.

While global topography models (GTMs) make sense for small irregular bodies, we often describe the local terrain using a digital elevation model (DEM). These DEMs may come from an SPG/SPC pipeline, or from somewhere else (e.g., a LIDAR during a lunar landing). Regardless of the source, these DEMs are often stored in an elevation image, where each pixel contains the elevation above/below a reference surface at a particular surface point. A very similar structure is created by a range image with a Flash LIDAR or a structured light sensor. These elevation (or range) images may be easily converted into quad meshes by connecting adjacent pixels, as is shown in Fig. 2.

The 3D shape of a celestial body is a critical component to answering important science questions. At the local level, 3D terrain models provide great insight into the local regolith properties, impact crater morphology, and various geological processes. At the global level, the gravity field of an irregularly shaped celestial body of constant density may be directly inferred from its polyhedral model. Although real bodies are not of constant density, the polyhedral gravity model has many advantages over the usual spherical harmonics (SH) approach for highly irregular bodies. This is especially important if the gravity model is to be used for navigation or mission design for a spacecraft trajectory that passes close to the body (and, perhaps, inside the SH circumscribing sphere).

The 3D shape/terrain model is also essential for navigation, especially for low-altitude orbits or for precision landing. Such terrain models are necessary to appropriately interpret range sensor (e.g., laser altimeter) measurements. They are also necessary for most common forms of TRN. For example, terrain models are necessary for registering a LIDAR-generated 3D point cloud to the local topography to localize the spacecraft during operations around a small body or during descent & landing. These terrain models are also necessary for rendering a predicted image for use in image-based TRN (either in orbit or during descent & landing).
within the vertex table and faces may be listed in any order within the facet table. Further, explicit connectivity allows the greatest amount of flexibility in mesh resolution, since it can support any integer number \( \geq 3 \) of vertices. The disadvantages are the memory required to store the facet table (which is generally larger than the vertex table) and the time required to find certain faces if they are unordered within the facet table.

Explicit connectivity also allows for the implicit bookkeeping of the surface normal of planar facets by ordering of the vertices describing each facet. Common convention is to order the vertices in a counterclockwise fashion when viewed from the outside of the mesh (although a clockwise convention is just as valid). Additionally, attribute tables allow for explicit bookkeeping of additional vertex attributes, such as colors, textures, and surface normals. Implicitly defined facet normals are useful for defining the directionality of a polygon, while explicitly defined vertex normals allow for smooth interpolation across a surface.

There are a number of common 3D mesh file formats that use explicit mesh connectivity, with PLY and OBJ being especially pervasive. These both assume a counterclockwise ordering of vertices for each facet to specify the facet’s normal direction. Of particular note for space science/engineering is the SPICE Digital Shape Kernel (DSK) Type 2 file format, which defines a triangular mesh using a vertex table and facet (what they call a “plate”) table.

Modern mesh processing pipelines (e.g., MeshLab) can easily handle very large meshes stored in these explicit file formats. Further, there are many benchmark models used for assessing mesh processing algorithms that contain millions of vertices or faces, such as the Stanford Dragon model (PLY with 566,098 vertices, 1,132,830 triangles), Stanford Lucy model (PLY with 14,027,872 vertices, 28,055,742 triangles) or the model of Michelangelo’s David (PLY with 940 million triangles). While still representative of the state-of-the-art, these examples are now over 10 years old. Similarly, the SPG-produced DEM of the Moon’s Taurus-Littrow Valley (the Apollo 17 landing site and its surroundings) is a 7,800 × 9,240 elevation map, corresponding to a quad mesh with 72,072,000 vertices and 72,054,961 faces. DEMs of comparable size and resolution may also be produced with SPC using the “bigmap” approach. Regardless of the source or object, manipulation of 3D meshes of this size does not generally happen in real-time, but may still occur on a timescale suitable for typical Earth-in-the-loop space science operations when appropriate conventions, software pipelines, and computational hardware are used.

It is sometimes necessary to either reduce the resolution of a mesh (sometimes called mesh simplification), as we may not always want to process millions of vertices or facets. In other cases we may wish to increase the mesh resolution (sometimes called subdivision). A vast amount of literature exists on performing both of these tasks for explicit meshes (both tri and quad). The interested reader is directed to good reviews on mesh simplification (e.g., decimation, compression as well as mesh subdivision). Many of these approaches are implemented in standard mesh processing software packages.

**Implicit Connectivity.** The mesh connectivity may be described implicitly through the ordering of vertices within the vertex table. Such an approach eliminates the need for the facet table. There are a variety of schemes one could use to implicitly describe vertex connection, but the implicitly connected quadrilateral (ICQ) format is the most pervasive in space science 3D modeling applications. This is largely due to its use within the Gaskell-derived SPC pipeline and its subsequent adoption in downstream tools. The prevalence of the ICQ format warrants some additional discussion here.

In the most straightforward case, consider the problem of generating a watertight GTM of an entire body. Achieving this task requires vertices around the entire body and a means for knowing which vertices are connected to one another. Thus, for the sake of defining

![Figure 4. The ICQ convention defines vertex connectivity using a cube with \( Q + 1 \) vertices along an edge (top). The cube may be unfolded in a particular fashion (bottom) such that the vertices on all the faces are indexed in the same direction. Each of the six faces has its origin in the upper left-hand corner and all use the \( i-j \) coordinate system shown below the unfolded cube. This example shows \( Q = 8 \).](http://graphics.stanford.edu/data/3Dscanrep/)
vertex connectivity, consider a grid of points on the face of a cube (Fig. 4). Let each cube face have \( Q + 1 \) vertices on an edge, so that each cube face has \((Q + 1)^2\) vertices and the entire cube has a total of \(6(Q + 1)^2\) vertices. It follows that each cube face has \(Q^2\) quadrilateral facets and that the entire model has \(6Q^2\) facets. The example illustrated in Fig. 4 is for \( Q = 8 \). The vertices on each face are indexed from 0 to \( Q \) with the origin in the upper left-hand corner of the face when the cube is unfolded in the manner shown in the bottom frame of Fig. 4. The quadrilateral connectivity of the vertices forming a facet is self-evident: clockwise from the upper left the indices forming a closed quad are \(\{i, j\}, \{i + 1, j\}, \{i + 1, j + 1\}, \{i, j + 1\}\), and then back to \(\{i, j\}\).

In order to generate watertight models using the ICQ format, the labels of redundant vertices along the edges and corners of each of the six faces must be replaced. Although there are \(6(Q + 1)^2\) labeled vertices in the entire ICQ model, only \(6Q^2 + 2\) vertices are independent. Redundant vertices are replaced according to the face hierarchy (from highest to lowest) of \(1 \& 6\), then \(2 \& 4\), and then \(3 \& 5\). When vertices belonging to two faces overlap on an edge/corner of the cube, the vertices belong to a higher hierarchy face are kept, while vertices belonging to a lower hierarchy face are replaced. This face hierarchy may be enforced by first replacing the top and bottom rows of faces 2 and 4 with the corresponding vertex labels in faces 1 and 6, then replacing the left and right rows of faces 3 and 5 with the corresponding vertex labels of faces 2 and 4, and lastly replacing the top and bottom rows of faces 3 and 5 with the corresponding vertex labels in faces 1 and 6. The correspondence between vertices in adjacent faces can be sometimes confusing due to the way indices are defined in the unfolded cube (Fig. 4). Therefore, to aid the reader (or programmer) in understanding how vertices in adjoining faces correspond to one another, we have provided diagrams centered about each of the four faces on which vertices are replaced in Fig. 5.

It should be emphasized that the cube structure described here is used to describe vertex connectivity only. The ICQ format is NOT the projection of points around a cube onto the body’s surface. Since the cube structure is used only for connectivity, it is possible to model non-convex surfaces that are not injective (one-to-one) to the surface of the cube (or a sphere). That is, the ICQ format supports 3D models of objects whose surface is multivalued along a radial direction. Furthermore, the density of vertices in any portion on the mesh is in no way related to the corresponding density of points on the cube from which the mesh vertices inherit their connectivity.

One of the principal disadvantages to the ICQ format is the limitation on possible mesh resolutions. Since \( Q \) must be a positive integer, the total number of mesh vertices grows according to \(6(Q + 1)^2\) — resulting in rather large steps in available mesh resolution as \( Q \) becomes large. Further, convention has led to the choice of \( Q \) being a power of two, as this permits easy mesh decimation. The largest ICQ GTM models available on PDS archives tend to be \( Q = 512\), which corresponds to 1,579,014 vertices and 1,572,864 quadrilateral facets. These ICQ models are modest in size compared to the state-of-the-art 3D models generated by laser scanning (e.g., the Lucy and David models referenced for meshes with explicit connectivity).

Due to the uniform structure of ICQ formats, mesh subdivision may be easily performed by adding additional nodes between each vertex and one in the center of a face. Conversely, mesh decimation may be achieved by simply deleting the appropriate vertices when \( Q \) is chosen to be a power of two. Both subdivision and decimation will maintain the connectivity between vertices due to the grid connectivity of the ICQ format. This can be seen in Fig. 6.

**Topology.** Visual artists and animators using 3D meshes are often concerned with the visual appeal of their models and how these models tell a story. It is important for the characters, objects, and environments to move naturally and have believable physics. Those using commercial 3D modeling software packages for processing space science data may encounter software conven-
Figure 6. Simplification of an ICQ mesh (left) may be performed by removing every other row and column of vertices (gray dashed line) when \( Q = 2^n \). Subdivision of an ICQ mesh (right) may be performed by adding new vertices (blue) between existing vertices (black), producing a mesh of resolution \( Q_{\text{new}} = 2Q_{\text{old}} \).

Figure 7. Example edge loops on a quad mesh of Phobos. This 3D model of Phobos was developed from Viking 1 & 2 images by Bob Gaskell using the stereophotoclinometry (SPC) pipeline. The PDS-archived 3D model is natively represented as a quad in the ICQ format.

The basic structure of the 3D art pipeline consists of five major steps: 3D modeling, UV unwrapping, texturing, rigging, and animating. In order for these to operate efficiently, the industry-wide standard is to model the 3D mesh using strictly quads. Quads allow models to be symmetric, have streamline topology, and can create smooth, organic shapes more easily, all of which are crucial for developing believable scenes.

Streamline topology is necessary to have good edge flow and therefore be able to represent more believable natural objects. This can be difficult to do with triangular faces, which can make the model topology very irregular. Good edge flow also allows a 3D artist to select an edge loop, which is a series of edges in a row that can be clicked on for ease of mesh manipulation. Edge loops on a 3D model of Phobos can be seen in Fig. 7. To add more detail to a mesh, edge loops can also be easily added since quads are able to subdivide nicely, as a quad simply divides into two perfectly even quads.

Meshes with changing topology can be used in a variety of simulation scenarios in which the mesh nodes need to break apart. A triangulated mesh broken between any two points still maintains its structure. A quadrilateral mesh cannot be broken between any two points like triangles can be, since this can cut across a quad diagonally and result in errors.

Rendering. Many rendering algorithms exist for generating synthetic images seen by a simulated camera viewing a digital 3D model. Which technique (or combination of techniques) is used depends on the computational resources and time available, the surface properties of the objects in the scene, and the desired photorealism of the final image. Most rendering techniques can be grouped into one of the following categories: rasterization, ray tracing, and radiosity.

Rasterization is the process of computing the mapping from 3D vector objects to a raster or pixel image. It is a key step in the graphics pipeline approach, which is used for real-time rendering applications such as video games or interactive graphic web programs. In OpenGL (one of the more common pipelines), the graphics pipeline is broken down into four main components: vertex shader, primitive assembler, rasterizer, and fragment shader. The vertex shader computes transformations (e.g., translations, rotations, scaling within the scene) and projections (e.g., from 3D scene coordinates to 2D window coordinates) of individual mesh vertices. The primitive assembler creates edges (lines) and faces (polygons) from the vertices. The rasterizer converts primitives into fragments, or segments of primitives that cover the area of a pixel. Lastly, the fragment shader determines the color of each fragment. Hidden surface algorithms (e.g., scan line rendering, painter’s algorithm, binary space partitioning trees) determine the order of overlapping fragments. Rasterization and the graphics pipeline approach are generally very fast compared to other methods. The result, however, is nonphysical, as these approaches only account for local lighting. Programs that allow a user to interactively view a 3D model, like those that a space scientist might use to visually inspect the shape of a celestial body or visualize a spacecraft trajectory, typically use a rasterization-based rendering approach.
Ray tracing is the process of tracing the path of a ray of light through a pixel in the image plane and simulating its interaction with a 3D scene of virtual objects. It is capable of producing a much higher degree of visual realism than rasterization or graphics pipeline approaches because it is able to account for a number of global lighting effects (particularly shadows, specular reflections, and refraction), but it has a much greater computational cost. It is better suited for applications that are compatible with long rendering times (e.g., visual effects for television and movies) or when physical results are required (e.g., scientific sensor simulations).

Radiosity is another method for computing global lighting effects, specifically diffuse inter-reflections. It leverages concepts from heat transfer in order to determine the amount of light that is reflected and radiated off of surfaces. It is well-suited for rendering images of building interiors which contain many diffuse reflecting surfaces. It provides a view-independent solution, which allows for fast display of arbitrary views.

Most real-time, rasterization-based rendering pipelines use triangular meshes, as modern GPUs have been optimized for this approach. This convention was chosen because it is usually faster to render tris than quads. One of the principal challenges lies with the planarity of the faces. The three vertices of a triangular facet are always on a single plane, thus every triangular face is necessarily planar. The four vertices of a quadrilateral face, however, are generally not coplanar—either due to floating point error (when one attempts to enforce facet planarity) or by letting the vertices freely float without regard to facet planarity (as is done in SPC). For these reasons, nearly all forms of rendering, from real-time rendering in game engines to 3D animation rendering, eventually break all polygons down into triangles.

Game development and real-time rendering require 3D meshes to look visually appealing but at the same time allow for fast, high quality performance. Upon importing any quad mesh into a game engine (e.g., Unity, Unreal Engine), it is common practice for the engine to automatically subdivide each quad into a pair of tris. This helps the engine render the mesh quickly and output complex, realistic images in real-time.

Ray Tracing. Ray tracing techniques are well suited for accurately simulating the physical behavior of light, making these methods a useful tool for a broad range of engineering applications. For example, ray tracing can be used to simulate optical sensors (e.g., altimeters, cameras, LIDARs) for mission design, as is shown pictorially in Fig. 8.

One of the first approaches to ray tracing, now known as ray casting, was presented by Arthur Appel in 1968. The method involves tracing rays (just one per pixel) from a viewer to the nearest object blocking its path.

A particular ray emanating from a sensor located at \( p_0 \) and traveling in a direction \( d \) (where \( |d| = 1 \)) may be parameterized according to

\[
p(t) = p_0 + td
\]

where \( t \geq 0 \) represents the scalar distance traveled along the ray’s path. For a single image, every ray shares the same camera origin \( p_0 \) but has a unique \( d \) described by its corresponding pixel’s unit vector direction. This geometry is shown pictorially in Fig. 9.

Ray Casting with Triangular Meshes. It is straightforward to analytically compute the intersection (if one exists) between a ray and a triangular facet. This makes ray casting with a triangular mesh fast and efficient. The classical approach for efficiently computing this intersection is the Möller-Trumbore Algorithm, which we briefly review here.

Consider a triangular facet, such as in Fig. 10, defined by the three vertices \( v_i, v_j, \) and \( v_k \). Define the edges

\[
e_{ik} = v_i - v_k \quad \text{and} \quad e_{jk} = v_j - v_k
\]

parameterized according to

\[
p(t) = p_0 + td
\]

where \( t \geq 0 \) represents the scalar distance traveled along the ray’s path. For a single image, every ray shares the same camera origin \( p_0 \) but has a unique \( d \) described by its corresponding pixel’s unit vector direction. This geometry is shown pictorially in Fig. 9.

Ray Tracing with Triangular Meshes. It is straightforward to analytically compute the intersection (if one exists) between a ray and a triangular facet. This makes ray casting with a triangular mesh fast and efficient. The classical approach for efficiently computing this intersection is the Möller-Trumbore Algorithm, which we briefly review here.

Consider a triangular facet, such as in Fig. 10, defined by the three vertices \( v_i, v_j, \) and \( v_k \). Define the edges

\[
e_{ik} = v_i - v_k \quad \text{and} \quad e_{jk} = v_j - v_k
\]
such that any point on the triangular surface relative to
the common vertex \( v_k \) is a linear combination of these
two basis vectors (forming the plane that passes through
all three vertices). That is,
\[
p(a, b) = v_k + a e_{ik} + b e_{jk}
\]
which we may rearrange to find
\[
p(a, b) = a v_i + b v_j + (1 - a - b) v_k
\]
Note, of course, that \( e_{ik} \) and \( e_{jk} \) are neither unit vec-
tors nor orthogonal, so they do not form an orthonormal
basis. Instead the 2D coordinates \( \{a, b\} \) in the plane of
the triangular facet are the so-called **barycentric coordi-
nates**. Conveniently, the 3D point \( p(a, b) \) is known to lie
within the triangular facet if \( a \geq 0, b \geq 0, \) and \( a + b \leq 1 \);
otherwise the point \( p(a, b) \) lies outside the triangular facet
(i.e., the intersection of the ray with the plane of the three
vertices is not interior to the triangle formed by these ver-
tices).

![Figure 10. Triangular facet with ray intersection at point \( p \).](image)

The Möller-Trumbore Algorithm takes advantage of
barycentric coordinates to quickly compute ray intersec-
tions with triangles.\(^\text{46}\) This algorithm equates the 1D
parametric ray equation (Eq. 11) to the description of
the triangular facet’s plane (Eq. 3) and evaluates the barycen-
tric coordinates \( \{a, b\} \) to see if the intersection conditions
are met. Therefore, observing that
\[
p_0 + td = v_k + a e_{ik} + b e_{jk}
\]
we may isolate the three scalar unknowns \( \{t, a, \) \) on
one side of the equation
\[
q = -td + a e_{ik} + b e_{jk}
\]
where \( q = (p_0 - v_k) \). Thus, defining the 3 \times 3 matrix
\[
A = [-d \ e_{ik} \ e_{jk}]
\]
we have the following linear system
\[
\begin{bmatrix}
t \\
a \\
b
\end{bmatrix}
= \begin{bmatrix}
q \\
-d \\
e_{ik} \\
e_{jk}
\end{bmatrix}
\]
where
\[
det(A) = (d \times e_{ik})^T e_{jk}
\]
Note, of course, that
\[
det(A) = (d \times e_{ik})^T e_{jk}
\]
These may be written separately as
\[
t = \frac{1}{\det(A)} (q \times e_{ik})^T e_{jk}
\]
\[
a = \frac{1}{\det(A)} (d \times e_{jk})^T q
\]
\[
b = \frac{1}{\det(A)} (q \times e_{ik})^T d
\]
In practice, one first computes \( a \) and \( b \) to check the
bounds for a triangle intersection: \( a \geq 0, b \geq 0, \) and
\( a + b \leq 1 \). If an intersection exists, then the distance \( t \)
may be calculated (if needed). This process can be seen
in Algorithm 1.

**Algorithm 1 Ray-Triangle Intersection**

1: **procedure** \([a, b, t] = \text{RTINTERSECT}(ray, tri)\) \( \triangleright \) Eq. 10
2: \quad compute \( \det(A) \) \( \triangleright \) escape algorithm
3: \quad if \( \det(A) < \epsilon \) then
4: \quad \quad false \( \triangleright \) escape algorithm
5: \quad else
6: \quad \quad calculate \( a \) \( \triangleright \) Eq. 12
7: \quad \quad if \( a \not\geq \) bounds then
8: \quad \quad \quad false \( \triangleright \) escape algorithm
9: \quad \quad else
10: \quad \quad \quad calculate \( b \) \( \triangleright \) Eq. 13
11: \quad \quad \quad if \( b \not\geq \) bounds then
12: \quad \quad \quad \quad false \( \triangleright \) escape algorithm
13: \quad \quad \quad else
14: \quad \quad \quad \quad calculate \( t \) \( \triangleright \) Eq. 11
15: \quad \quad return \( a, b, t \) \( \triangleright \) intersection

**Ray Casting with Quadrilateral Meshes.** Consider a
quadrilateral face defined by the four vertices \( v_i, v_j, v_k, \) \( v_l \).
These vertices are often not-coplanar, as is the
case in SPC-developed ICQ meshes and many other com-
monly encountered formats. In these cases, it is common
to describe the surface within the quadrilateral face as the
bilinear interpolation of its four vertices. We may param-
eterize the bilinear surface for a quadrilateral face by the
2D coordinates \( \{m, n\}\):\(^\text{47}\)
\[
p(m, n) = (1 - m)(1 - n)v_k + (1 - m)nv_j + m(1 - n)v_l + mnv_h
\]
where a point is within the quadrilateral face when \( 0 \leq
\{m, n\} \leq 1 \). This is shown graphically in Fig. 14.
Following a similar line of reasoning as for a triangular face, we may equate the 1D parametric ray equation (Eq. 1) to the description of the quadrilateral face’s bilinear surface (Eq. 14) and evaluate the coordinates \( \{m, n\} \) to see if the intersection conditions are met. Therefore, observing that

\[
p_0 + td = (1-m)(1-n)v_k + (1-m)mv_j + m(1-n)v_i + mnv_h
\]

we may isolate the three scalar unknowns \( (t, m, n) \) on one side of the equation

\[
q = -td + me_{ik} + ne_{jk} + mn\alpha
\]

where we have simplified the known vectors as

\[
a = v_k - v_j - v_i + v_h
\]
\[
e_{ik} = v_k - v_i
\]
\[
e_{jk} = v_j - v_k
\]
\[
q = p_0 - v_k
\]

We immediately see that Eq. 16 is no longer linear in the unknowns (due to the quadratic \( mn \) term), such that this cannot be written as a simple linear system. Further, due to the quadratic nature of Eq. 16, it follows that a ray may intersect a general quadrilateral face either zero, one, or two times. This may be seen in Fig. 12.

Notice the similar structure between Eq. 6 of triangular ray casting with Eq. 16 of quad ray casting. Equation 16 simply has an additional quadratic term \( mn\alpha \).

Fortunately, however, we observe that Eq. 16 is linear in \( t \), and we may exploit this fact to find an analytic solution. This solution follows the approach suggested by Ramsey, Potter, and Hansen.

Proceed by rewriting Eq. 16 as the three scalar equations (for each dimension) described by the vector equation and rearrange each to solve for \( t \),

\[
t = (me_{ik} + ne_{jk} + mnax - q_k)/dz
\]
\[
t = (me_{ik} + ne_{jk} + mnay - q_y)/dy
\]
\[
t = (me_{ik} + ne_{jk} + mnaz - q_z)/dz
\]

Setting \( t \) for the \( x \) and \( y \) directions equal to the \( z \) direction results in two equations with two unknowns (\( m \) and \( n \)).

Solving for \( m \) in one equation and substituting it back into the remaining equation yields

\[
0 = (A_2C_1 - A_1C_2)n^2 + (A_2D_1 - A_1D_2 + B_2C_1 - B_1C_2)n + (B_2D_1 - B_1D_2)
\]

where

\[
A_1 = a_xdz - a_xdx
\]
\[
A_2 = a_ydz - a_ydy
\]
\[
B_1 = e_{ik}dz - e_{ik}dx
\]
\[
B_2 = e_{ik}dy - e_{ik}dz
\]
\[
C_1 = e_{jk}dz - e_{jk}dx
\]
\[
C_2 = e_{jk}dy - e_{jk}dz
\]
\[
D_1 = q_xdx - q_xdz
\]
\[
D_2 = q_ydy - q_ydz
\]

Finding the roots of the quadratic equation in Eq. 18 provides the possible values for \( n \). For any given \( n \), one may solve for the corresponding \( m \)

\[
m = \frac{n(C_1 - C_2) + (D_1 - D_2)}{n(A_2 - A_1) + (B_2 - B_1)}
\]

The ray intersects the quadrilateral patch when both \( 0 \leq m \leq 1 \) and \( 0 \leq n \leq 1 \). In this case, one may compute the value of \( t \) (if desired) by evaluation of Eq. 17. When more than one intersection exists, the point corresponding to the smaller value of \( t \) should be used—since this is the point that would be seen by the camera. This process can be seen in Algorithms 2 and 3.

Brief Remarks on Ray Casting with other Explicit Surface Representations. Recent work has considered the use of other explicit surfaces for the 3D modeling of celestial bodies, particularly NURBS and Bézier surfaces. Readers interested in ray tracing with these surfaces are directed to the substantial body of literature on the topic.

Converting Between Formats. It is often necessary to convert between triangular and quadrilateral meshes.
Algorithm 2 Ray-Quad Intersection

1: procedure \([m, n, t] = \text{RQIntersect}(\text{ray}, \text{quad})\)
2: \(r = \text{solve for } n\) \hspace{1em} \(\triangleright\) Eq. [18]
3: if \(r = 0\) then
4: \(n = \text{false}\) \hspace{1em} \(\triangleright\) escape algorithm
5: if \(r = 1\) then
6: \([m, n, t] = \text{Calculate}(n)\) \hspace{1em} \(\triangleright\) Algorithm 3
7: else if \(n = \text{false}\) then
8: \(n = \text{false}\) \hspace{1em} \(\triangleright\) escape algorithm
9: if \(n_1 = \text{false} \&\& n_2 = \text{false}\) then
10: \(n = \text{false}\) \hspace{1em} \(\triangleright\) escape algorithm
11: if \(n_1 = \text{false}\) then
12: \(m, n, t\) \hspace{1em} \(\triangleright\) intersection
13: if \(n_2 = \text{false}\) then
14: \(m, n, t\) \hspace{1em} \(\triangleright\) intersection
15: \(m_1, n_1, t_1\) \hspace{1em} \(\triangleright\) intersection
16: else if \(t_1 < t_2\) then
17: \(m_1, n_1, t_1\) \hspace{1em} \(\triangleright\) intersection
18: \(m_2, n_2, t_2\) \hspace{1em} \(\triangleright\) intersection

Algorithm 3 Calculate Intersection Parameters

1: procedure \([m, n, t] = \text{Calculate}(\text{ray}, \text{quad}, n)\)
2: given \(n\) \hspace{1em} \(\triangleright\) Eq. [18]
3: if \(n \notin \text{bounds}\) then
4: \(n = \text{false}\)
5: else
6: \(m = \text{calculate } m\) \hspace{1em} \(\triangleright\) Eq. [19]
7: if \(m \notin \text{bounds}\) then
8: \(n = \text{false}\)
9: else
10: \(t = \text{calculate } t\) \hspace{1em} \(\triangleright\) Eq. [17]
11: if \(t < 0\) then
12: \(n = \text{false}\)
13: else
14: \(m, n, t\)

as the native model format may not match the requirements of a particular software pipeline. Many rendering pipelines (e.g., Blender: https://www.blender.org) and mesh processing pipelines (e.g., MeshLab[22]) contain simple routines to achieve this task.

Conversion of a quad mesh to tri mesh simply requires that each quad be split into two triangles. The difficulty lies in the scheme that one uses to decide between the two possible ways of splitting each quad within the mesh. There are a variety of schemes, ranging from simple (and fast) to complicated (and slow). The simplest method is to choose the vertices to connect a priori, such that quads are always split by connecting the first and third vertex or by connecting the second and fourth vertex. A slightly more expensive approach is to always split the quadrilateral facet by its shortest diagonal (this is the method suggested for triangulating SPC-produced quads in the documentation of [5]). At even greater expense, each facet may be split using the diagonal that creates the largest triangle.

The conversion from a tri mesh to a quad mesh is less straightforward, as arbitrary removal of a shared edge to combine two triangles into a quadrilateral leads to non-planar facets in general, can create non-convex quadrilateral facets, and can leave stray triangles that need to be cleaned up. A multitude of methods have been suggested for addressing these challenges.[33][51][54]

Conclusion. This work briefly reviews the advantages and disadvantages of modeling the 3D shape of celestial bodies using triangular or quadrilateral meshes. We discuss common uses for such models as well as the most pervasive model formats (PLY, OBJ, DSK, and ICQ). Considerations regarding tool pipelines, file size, topology, rendering, and sensor simulation are discussed. Efficient algorithms for ray tracing are presented for both triangular and quadrilateral meshes — and they are presented in a way that highlights their similar mathematical structure.

Acknowledgments. The authors thank William Hicks and Shawn Lawson of Rensselaer Polytechnic Institute for their insights on mesh modeling.

References.


