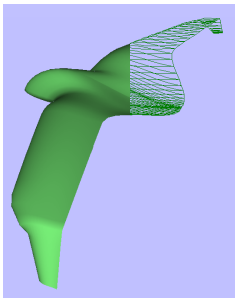
Trigons, Normals, and Quadragons

Vector-based Solutions to the Classic Problems of Computer Graphics



Presented at 2007 ESA Western Workshop

J. Philip Barnes

A computer graphic (CG) image displays a three-dimensional (3D) object, as realistically as possible, on a 2D computer screen. In this presentation, we will demonstrate 3D geometry math modeling and visualization, including rendering at the individual pixel level. In so doing, we will introduce new alternatives to both classic and contemporary solutions to the classic problems of computer graphics. All of the “alternate” solutions herein are new. Whereas some offer additional insight and complement contemporary methods, others offer either increased computational efficiency or increased rendering accuracy.

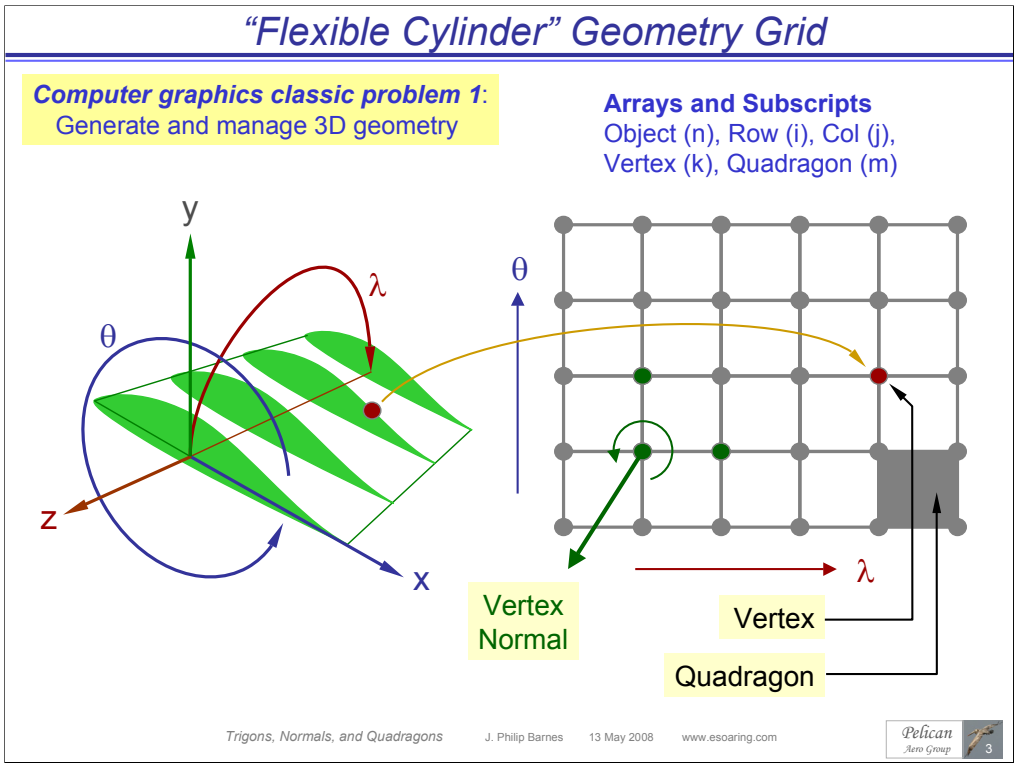
Overview

Display a 3D-math-modeled shape on a 2D screen
Introduce new solutions to several classic problems

Presentation Contents

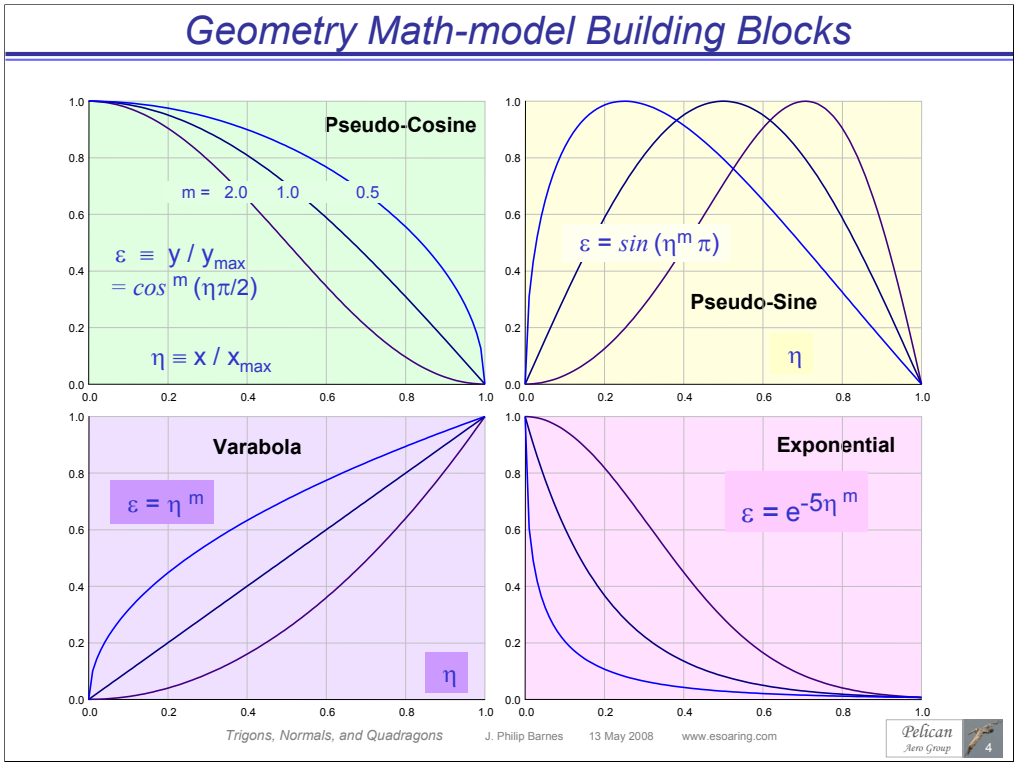
- "Flexible-cylinder" geometry grid
- Geometry math modeling tools
- Vectors ~ history & key operations
- Fast world-to-screen transformation
- Review & Supplement: Z-buffer algorithm
- Vector solution : "Point-in-polygon"
- Vector solution: Inter-vertex interpolation
- "Pyrometer" ambient lighting model
- Apply all herein ~ "TrigonoSoar"

This chart has no footnotes.



A three-dimensional surface, whether for example a wing, toriod, or sphere, can be mapped as a flexible "distorted cylinder." To "render" the object as a CG image, we will approximate its surface as an arrangement of "quadragons," each having ranges of "latitude (λ)" and "longitude (θ)," and each having vertex *normal vectors* pointing away from the surface.

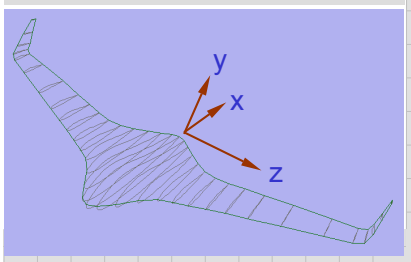
With a system of arrays and subscripts thereof, we can keep track of quadragons, vertices, and normal vectors. Then, once we find ourselves more deeply involved in our CG calculations, we discover that we'll need to determine the applicable trigon for each quadragon at hand, whereby the trigon will "inherit" many properties of its "parent." Toward this end, the "type" declaration of Visual Basic proves invaluable. Herein, our graphics are rendered using only the VB "set pixel color" command. Our geometry definition and vector operations will consistently use the right-hand rule.

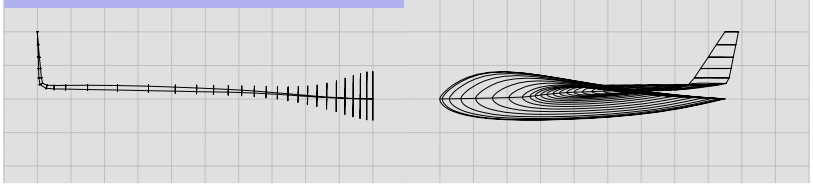


Here we plot several simple but powerful functions which can be used to construct building blocks for the 2D or 3D geometry of streamlined objects. In most cases, these functions have a normalized maximum “length” and/or “height,” each of which can be scaled to model the physical dimension at hand. In addition, these functions incorporate a variable exponent enabling shape adjustment, and more than one function can be added to build shapes of increasing complexity. For example, drawing from the modified trigonometric functions, the “pseudo-sine” with an exponent at or near 0.5 can be used to model the basic half thickness of an airfoil, with maximum “x” representing airfoil chord, and maximum “y” representing airfoil half thickness. Then, an additional smaller term, perhaps using a different function and/or exponent, can model a localized bump.

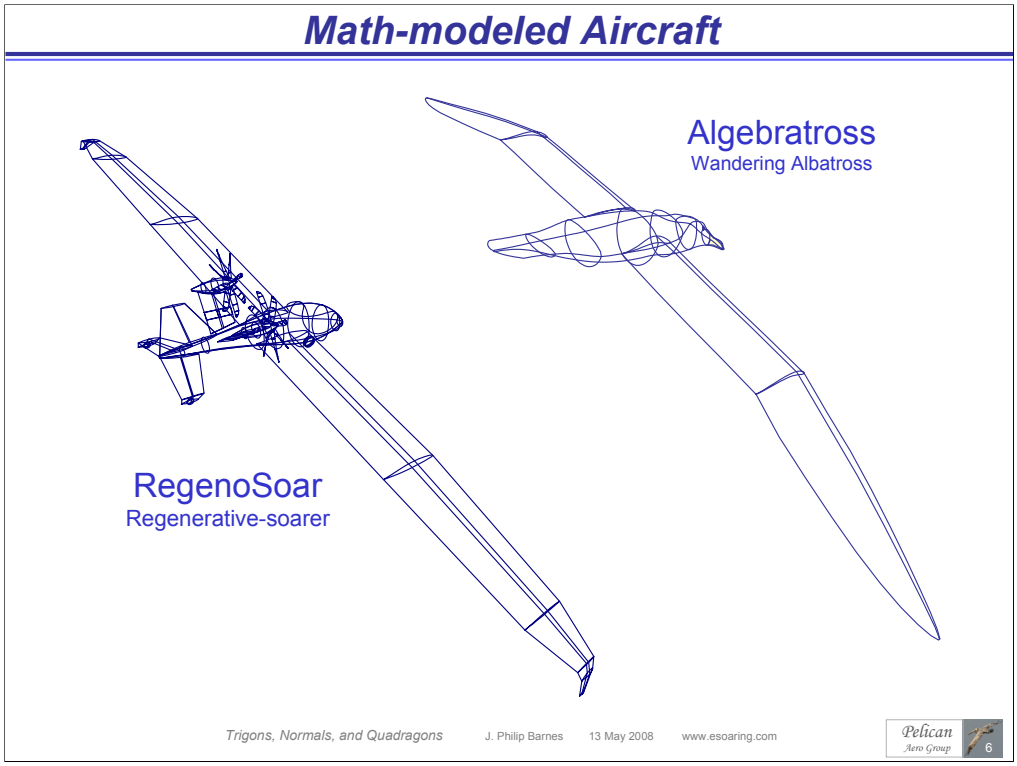
"TrigonoSoar" Math-modeled 3D Surface

- Trigonometric Math Modeled
- To be rendered with trigons
- Chord, thickness, twist, etc. parametric along the spar
  from centerline to winglet tip





We introduce here "TrigonoSoar," a math-modeled object to which we will apply our new rendering methods later herein. TrigonoSoar is a blended wing with numerous geometrical parameters (such as backbone, chord, thickness, camber, and washout) modeled parametrically with 2D (y-z-planar) distance along the spar from centerline to winglet tip. Implementations of various modified trigonometric functions for geometry definition are evident in the plan and profile views.

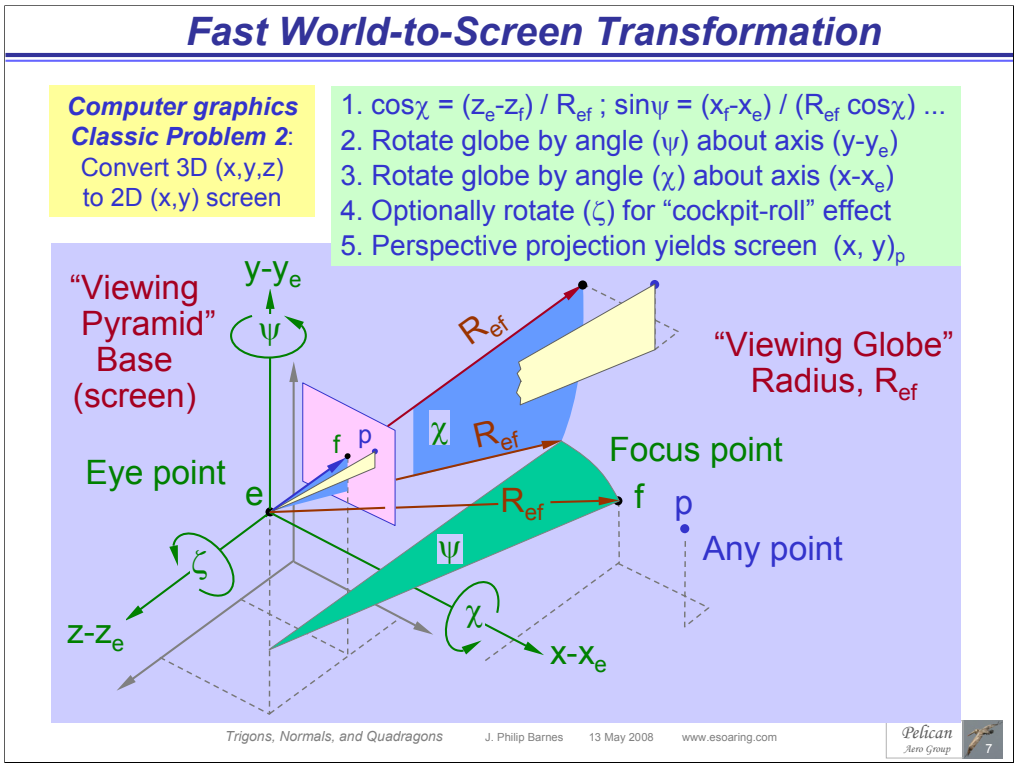


Here we illustrate more complex applications of the previously-described geometry building blocks. Both “Algebratross” and “RegenoSoar” are fully math modeled with such tools.

“Algebratross” is a model of the wandering albatross, which uses its 3.5-m wingspan and dynamic soaring technique to remain aloft indefinitely over a waveless sea, without flapping its wings.

Whereas the albatross uses the vertical gradient of horizontal wind speed to remain aloft, the “RegenoSoar” regenerative-soaring aircraft remains aloft, potentially indefinitely at high altitudes, by using its propeller (having symmetrical blade sections) as a turbine to recharge stored energy whenever an updraft is encountered.

As we next progress from “wireframe” to “rendered” computer graphics, we enter the world of “vectors” and “algorithms.”

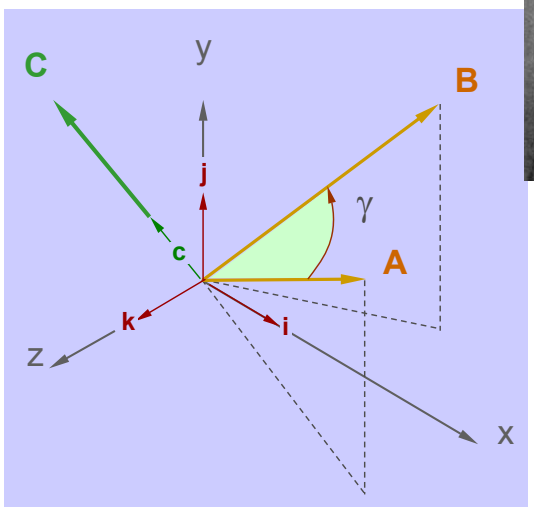


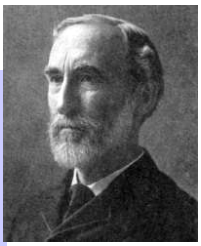
We can imagine a 3D surface as a collection of "points" in space. This "second" classic problem CG is to place a given point (p) on the screen, given the coordinates of the eye point (e), focus point (f), and the point (p) itself. This task is greatly simplified if we imagine the observer to see the world through a hole in the apex of a "viewing pyramid" turned on its side. The viewing pyramid is of course a well-known idea in the world of CG, but our intent here is to apply the viewing pyramid concept with greater efficiency.

Let's say the viewer initially looks due east, but now intends to focus on some point (f) which resides to the southeast at some different elevation. The observer could "pan" and then "pitch" the pyramid until the point (f) were centered on the "screen" (pyramid base). However, an easier alternative (mathematically) is to leave the viewing pyramid fixed as the "viewing globe" of radius (Ref), centered at (e), is twice rotated for the same result.

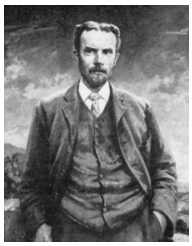
Relative to the standard "***world-to-screen transformation***" methods, this approach requires just two rotations, and no translations. Key to the method is carrying out such rotations about axes passing through the point (e), whereby the rotations operate on the coordinate differences ($x-x_e$, $y-y_e$, $z-z_e$), not the (x,y,z) coordinates themselves. The third rotation is optional for "cockpit roll."

Vectors ~ Origin and Key Operations





J. Willard Gibbs

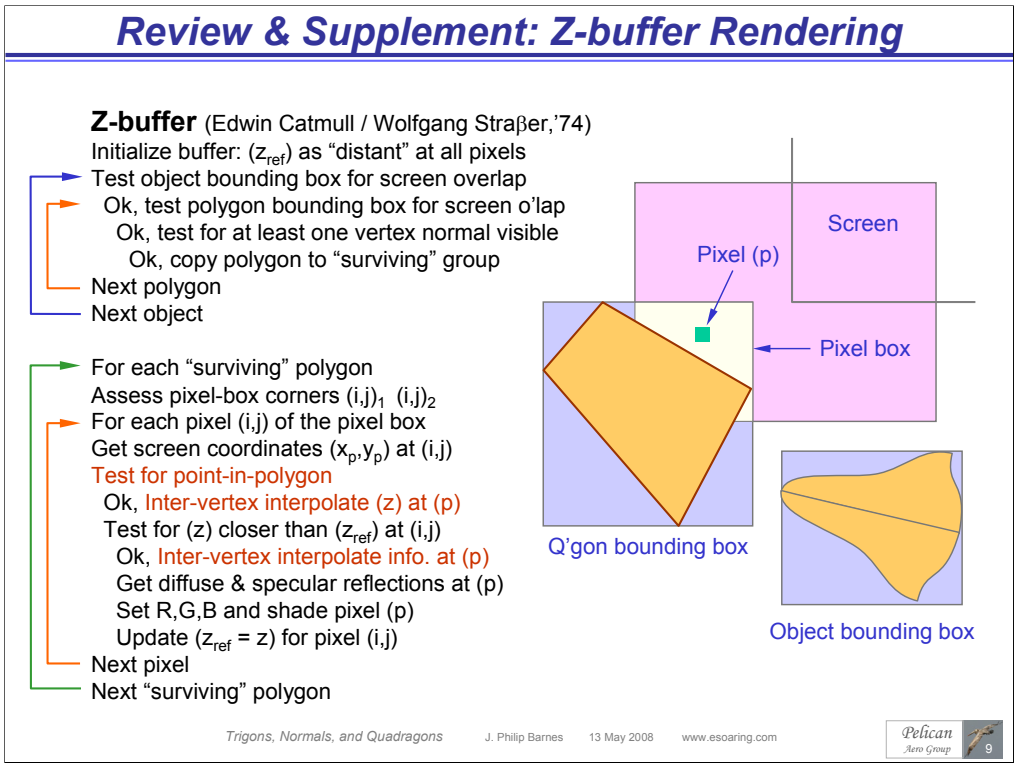


Oliver Heaviside

Unit vectors
$\mathbf{a} \equiv \mathbf{A} / |\mathbf{A}|$
$\mathbf{b} \equiv \mathbf{B} / |\mathbf{B}|$
$\mathbf{c} \equiv \mathbf{C} / |\mathbf{C}|$
$\cos \gamma = \mathbf{a} \cdot \mathbf{b}$
$\sin \gamma = |\mathbf{a} \times \mathbf{b}|$

We are immensely indebted to both Willard Gibbs and Oliver Heaviside, at the very least for their independent invention of vectors, which took place in the 19th century. However, both made many other significant contributions to math and science. Gibbs is perhaps best known for his analysis of available energy in thermodynamic and chemical processes. Heaviside took the original twenty equations of electro-magnetism published by Maxwell, and with the aid of new-found vectors and other powerful tools, condensed "Maxwell's Equations" into just four.

Armed with this most powerful tool known as "vectors," our computer graphics methods herein will make good use of the dot product, cross product, tip-to-tail addition, and scaling vector operations.



Here we provide a concise summary of the well-known "Z-buffer" rendering algorithm, while adding important supplementary steps. Along the way, we highlight in red those aspects of the algorithm which will make use of the newer methods herein, and point out where computational efficiency is most important. Although the z-buffer algorithm is carried out (after the world-to-screen transformation) in 2D screen coordinates, the goal of inter-vertex interpolation is to obtain the necessary 3D information for each applicable pixel.

Ultimately, for every polygon fully or partially displayed on the screen, we must test every applicable pixel. In working our way from entire objects down to the pixel level, we first take the opportunity to avoid the myriad calculations for any object having a "bounding box" that doesn't overlap the screen. The same exclusion test is applied to quadragons, ultimately yielding an applicable "pixel box," within which we then test for "point in polygon." Upon passing all tests, we are ready to interpolate key properties (i.e., z-coordinate), at the pixel (p), given the properties at the three vertices of the applicable trigon for the quadragon at hand. Finally, we compute the diffuse and specular reflections at the pixel and assign a pixel color.

Clearly, we should strive for the computational efficiency, particularly if a new image is to be displayed on the screen at perhaps 100 times per second, and/or when a computation resides in the innermost loop of the algorithm. Such is the case for both the "point-in-polygon" test and "inter-vertex interpolation." We next propose efficient, vector-based solutions for these two classic problems.

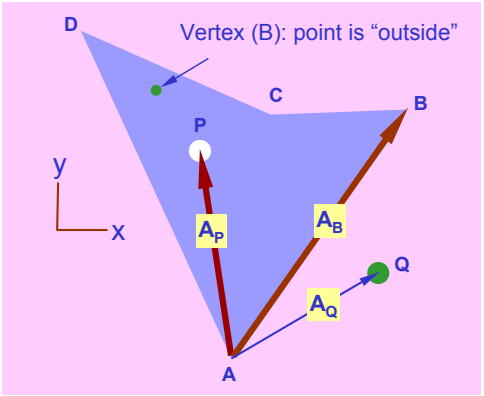
"Point-in-Polygon" Problem ~ Vector Solution

Computer graphics classic problem 3:
After the world-to-screen transformation,
is the point (P) inside the polygon?

Classic: Algebraic or trigonometric
Contemporary: *Barycentric Coords*.
Complementary alternative: Vectors

Unit vector (**k**) points out of screen
Designate any vertex as (A), with:
Vector ($\mathbf{A_B}$) to the next vertex (B)
Vector ($\mathbf{A_P}$) to the point (P) inside
Vector ($\mathbf{A_Q}$) to the point (Q) outside

$\mathbf{A_B} \times \mathbf{A_P}$ points out of the screen
$\mathbf{A_B} \times \mathbf{A_Q}$ points into the screen
Eureka!



The point (P) resides outside if, for *any* trigon vertex, or,
[Postulate] *at least n-2 "n-gon"* vertices, $(\mathbf{A_B} \times \mathbf{A_P}) \bullet \mathbf{k} < 0$

The classic solutions to the classic "Point-in-Polygon" problem often take a cumbersome algebraic form, or a trigonometric form which requires computations at all three or four vertices. The contemporary solution, which can often deem the point "outside" after interrogating just one vertex, makes use of *barycentric coordinates* (BC). For the point (p), the BC method sequentially computes up to three barycentric coordinates (α, β, and $\gamma = 1-\alpha-\beta$), and makes a quick exit, deeming the point (p) as "outside" upon finding any barycentric coordinate to be negative. Likewise, the new and alternate vector-based method above quickly "bows out" if the simple vector operation shown yields a negative scalar result.

We postulate the vector-based method as applicable to any "n-gon" whereupon (n-2) vertices yield the "point outside" result. The author learned of the BC approach only after this vector-based method was developed.

Inter-vertex Interpolation ~ Vector Solution

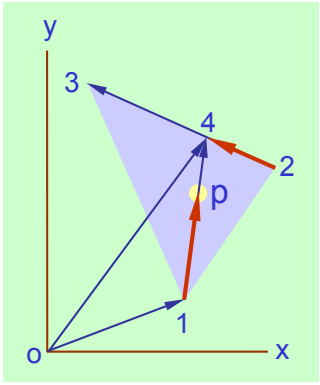
Computer graphics classic problem 4:
Given geometry & optics at the vertices,
interpolate the values thereof at point (p)

Classic method: "scan line" (comp. intensive)
Contemporary method: *Barycentric Coords*.
Complementary Alternative : Vectors

Apply vector tip-to-tail addition & scaling
Solve for & store two "position parameters"

$$\mathbf{R}_{o4} = \mathbf{R}_{o1} + \mathbf{R}_{14} = \mathbf{R}_{o1} + \alpha\,\mathbf{R}_{1p}$$
$$= \mathbf{R}_{o2} + \mathbf{R}_{24} = \mathbf{R}_{o2} + \beta\,\mathbf{R}_{23}$$

Equate $\mathbf{R}_{o4}$ (Δx) & (Δy) to get α, β



$$\delta \equiv (x_p - x_1)(y_2 - y_3) - (y_p - y_1)(x_2 - x_3)$$
$$\alpha = \left[(x_2 - x_1)(y_2 - y_3) - (y_2 - y_1)(x_2 - x_3) \right] / \delta$$
$$\beta = \left[(x_p - x_1)(y_2 - y_1) - (y_p - y_1)(x_2 - x_1) \right] / \delta$$

Finally, interpolate at (p):
$z_p = z_1 + [z_2 + \beta\,(z_3 - z_2) - z_1] / \alpha$

For polygon sorting, shading, and related operations, we need to interpolate between trigon vertices to determine the properties at a point (p) within the trigon. Relative to the traditional "scan-line" method, which must compute various *distances*, our new vector-based solution computes various *differences*. Only the latter avoid computationally-intensive square-root operations.

The figure above illustrates the vector approach to obtain any property (such as z-coordinate or light intensity) at the point (p), given that same property at each of the three vertices of the trigon. With the aid of tip-to-tail vector addition and scaling, the method solves "ahead of time" two simultaneous equations to yield two "location parameters" (α,β) which apply to the point (p) for the applicable trigon.

Those familiar with the contemporary approach (barycentric coordinates) will note both similarities to, and differences from, the formulas shown above. Although the BC method is perhaps more elegant, the vector-based method provides additional insight and illustrates the unity of mathematics, whereby totally different methods yield the same result. Again, the author learned of the BC method only after independently developing this vector-based approach.

"Pyrometer Model" of Ambient Lighting

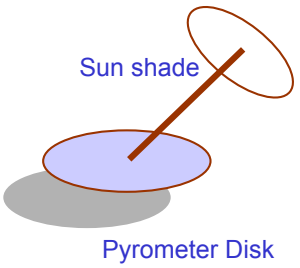
Computer graphics classic problem 5:
Model ambient light *before* illumination

Classic lighting model: Uniform ambient
Then add illumination

Problem:
"Grey silhouette" *until* lights are added
Poor foundation of total reflection model

Idea:
Vary ambient reflection with orientation
~ good visualization *before* illumination

Implementation:
Pyrometer measures skylight intensity
Rotated on it side ~ half the intensity
Inverted ~ zero intensity (@ albedo=0)



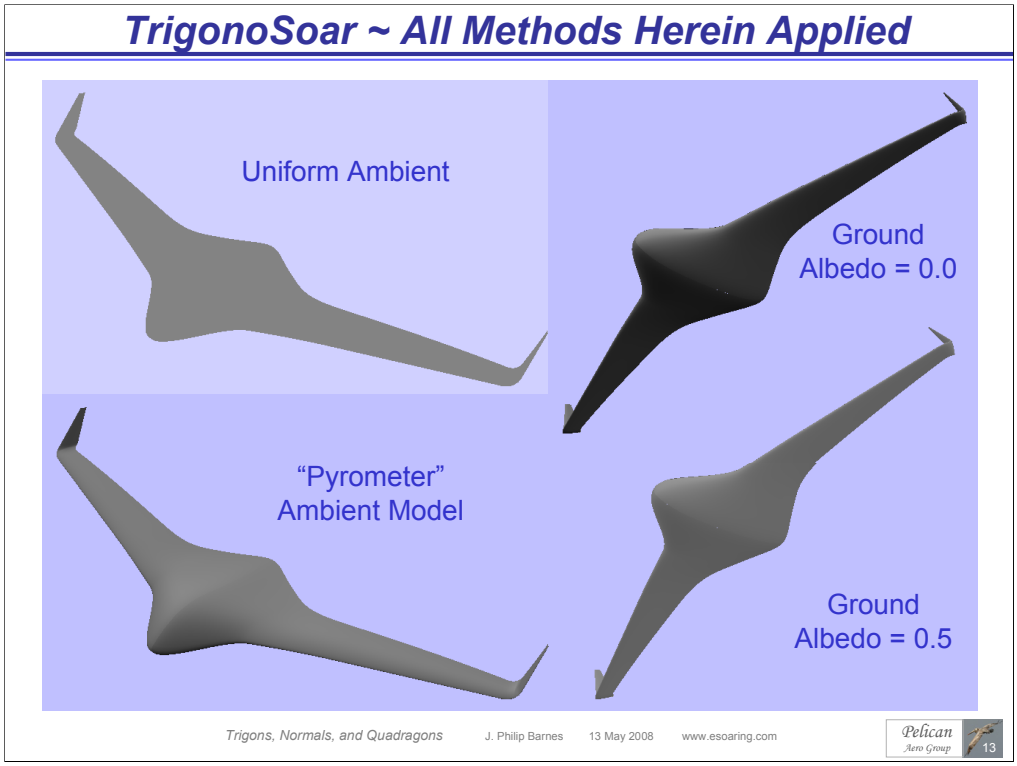
Define:
Pyrometer tilt angle (τ)
Ground albedo (α)

Ambient Reflectance
$\rho = 1 - \tau/\pi + \alpha\ \tau/\pi$

We have now arrived at the point where we are ready to "shade the pixel." The established shading models, including some which are quite complex, usually begin with an "ambient" light intensity before illuminating the scene with one or more lights. However, such modeling depends on illumination for surface definition. Prior to illumination, the uniform ambient model applies a uniform "shade of gray" to the object on the screen. Since our objective is to most accurately simulate all reflections acting together, it would appear that "fixed ambient" modeling lays a poor foundation for subsequent illumination.

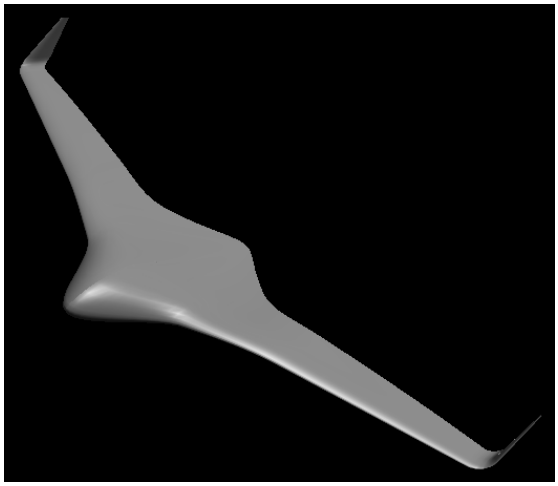
Thus, here we intentionally delay illumination until a more satisfactory ambient lighting model is in place. In so doing, we introduce the scientific instrument known as a "pyrometer," used to measure the intensity of diffuse skylight. The pyrometer, kept shielded from the sun, would register half the normal intensity if turned on its side (assuming for the moment a ground albedo of zero). If inverted, the pyrometer would register zero light intensity, again assuming zero ground albedo.

For our "pyrometer" ambient lighting model, we imagine each quadragon (or local point therein) to be a tiny pyrometer. If the local normal vector points "up," the pixel will reflect the maximum ambient light intensity. Ground albedo is adjustable in the model, which although quite simple, nonetheless represents a major improvement over fixed ambient lighting. This is apparent in the next slide, where we render "TrigonoSoar" with only ambient light. Note that such ambient light reflection will, and should, remain the same regardless of the position of lights, once such lights are added.



This slide has no footnotes.

TrigonoSoar ~ Standard Illumination Added



This slide has no footnotes.

Summary ~ *Trigons, Normals, and Quadragons*

New Solutions to several classic problems of CG

Complementary, faster, or better

- World-to-screen transformation ~ just two rotations
- Point-in-polygon ~ vector solution with “quick exit”
- Interpolation at point (p) ~ *differences*, not *distances*
- “Pyrometer” ambient ~ do best *before* illumination

About the Author

Phil Barnes has a Bachelor's Degree in Mechanical Engineering from the University of Arizona and a Master's Degree in Aerospace Engineering from Cal Poly Pomona. He has 28-years of experience in performance analysis and computer modeling of aerospace vehicles, engines, and subsystems, primarily at Northrop Grumman. He has authored SAE technical papers on aerodynamics, dynamic soaring, and regenerative soaring. This latest presentation brings together Phil's knowledge and passions for computer graphics, geometry math modeling, and soaring flight.

Email: PelicanAG@AOL.com