Hughes et al. 2013. CG Principles and Practice. ISBN-13: 978-0321399526. Addison-Wesley Professional. 3 edition (July 20, 2013)

List of Principles

Here we collect the principles that we explicitly espoused throughout the book. Like koans, these are perhaps best used to reflect on what you have learned, as new apertures through which to view those topics, and as a concise way of discussing the patterns that underlie computer graphics. Simply studying them outside of their context would not be effective.

We view this book as a series of case studies of concrete techniques—the *practice* that shows how to apply the principles to image synthesis and other problems. Our hope is that after reading many of the chapters you will agree that there is indeed a compact set of ideas and many ways of applying them.

As computer graphics scientists and engineers, we explore a vast sea of knowledge into which mathematics, engineering, physics, biology, psychology, and art flow. These principles chart the many courses of "computer graphics" that allow us to appreciate those disciplines and make forward progress in our own. For example, dynamics simulation of a gear system and estimating light transport are two of the many destinations presented in the field, but you can reach them with the same mathematical vehicles by following parallel courses of numerical integration techniques.

Computer graphics practice vertically integrates techniques from concrete, low-level engineering to abstract, high-level mathematics. A rendering engineer at a game company must not only understand mathematics and physics, but also must have reasonable computer organization skills to understand a graphics processor, cache, and bandwidth. That same engineer must also follow good software engineering methodology to work in a team of other programmers and artists, and nearly every day relies on calculus, geometry, optics, and color theory. Our principles vary in scope accordingly, from implementation strategies to modes of thought.

Know Your Problem principle: Know what problem you are solving.

Approximate the Solution principle: Approximate the solution, not the problem.

Wise Modeling principle: When modeling a phenomenon, understand the phenomenon you're modeling and your goal in modeling it, *then* choose a rich-enough abstraction, and *then* choose adequate representations to capture

your abstraction within the bounds of your resources, and finally, *test* to verify that your abstraction was appropriate.

Visual System Impact principle: Consider the impact of the human visual system on your problem and its models.

Coordinate-System/Basis principle: Always choose a coordinate system or basis in which your work is most convenient, and use transformations to relate different coordinate systems or bases.

First Pixel principle: The first pixel is the hardest.

Visual Debugging principle: Use visual displays to help you debug and understand your graphics programs.

Hierarchical Modeling principle: Whenever possible, construct models hierarchically. Try to make the modeling hierarchy correspond to a functional hierarchy for ease of animation.

Implementation principle: If you understand a mathematical process well enough, you can write a program that executes it.

Parametric/Implicit Duality principle: There's a duality between parametric and implicit forms for shapes: In general, it's easy to find an intersection between shapes where one's described implicitly and the other parametrically, and harder when either both are implicit or both parametric.

Tilting principle: If T' is an oriented triangle in plane P' with normal $\mathbf{n}'$, and T is its projection to plane P, the projection being along the unit normal $\mathbf{n}$ to P, then the signed area of T is $\mathbf{n}' \cdot \mathbf{n}$ times the signed area of T'.

Division of Modeling principle: Separate the mathematical and/or physical model of a phenomenon from the numerical model used to represent it.

Meaning principle: For every number that appears in a graphics program, you need to know the semantics, the **meaning,** of that number.

Transformation Uniqueness principle: For each class of transformations—linear, affine, projective—and any corresponding coordinate frame, and any set of corresponding target elements, there's a unique transformation mapping the frame elements to the correponding elements in the target frame. If the target elements themselves constitute a frame, then the transformation is invertible.

High-Level Design principle: Start from the broadest possible view. Elements of a graphics system don't separate as cleanly as we might like; you can't design the ideal representation for an emitter without considering its impact on light transport. Investing time at the high level lets us avoid the drawbacks of committing, even if it defers gratification.

Noncommutativity principle: The order of operations often matters in graphics. Swapping the order of operations can introduce both efficiencies in computations and errors in results. You should be sure that you know when you're doing so.

API principle: Design APIs from the perspective of the programmer who will use them, not that of the programmer who will implement them or from the mathematical notation used in their derivation. For example, a single BSDF $f(\omega_i, \omega_o)$ mapped to a function API `Color3 bsdf(Vector3 wi, Vector3 wo)` is easy to implement but hard to use in a real renderer.

Early Optimization principle: It's worth optimizing early if it makes the difference between an interactive program and one that takes several minutes to execute. Shortening the debugging cycle and supporting interactive testing are worth the extra effort.

Level of Detail principle: Level of detail is important for both efficiency *and* correctness.

Average Height principle: The average height of a point on the upper hemisphere of the unit sphere is $\frac{1}{2}$. Thus, for any unit vector $\mathbf{n}$, the integral

$$\int_{\{\omega \in \mathbf{S}^2 : \omega \cdot \mathbf{n} \geq 0\}} \omega \cdot \mathbf{n} \, d\omega = \pi.$$

Structure principle: Treat surprising structural symmetries and asymmetries as both clues about underlying structure and as warnings to check the robustness of any plan. For example, if otherwise similar elements differ by orders of magnitude or demand different parameters, as is the case for the fundamental forces of nature, something interesting is going on that can either lead to insight if followed, or bite you if ignored.

Culling principle: It is often efficient to approach a problem with one or more fast and conservative solutions that narrow the space by culling obviously incorrect values, and a slow but exact solution that then needs only to consider the fewer remaining possibilities.

Design Tradeoff principle: The art of architecture design includes identifying conflicts between the interests of implementors and users, and making the best tradeoffs.

Memory principle: The primary challenge of memory is coping with access latency and limited bandwidth. Capacity is a secondary concern.