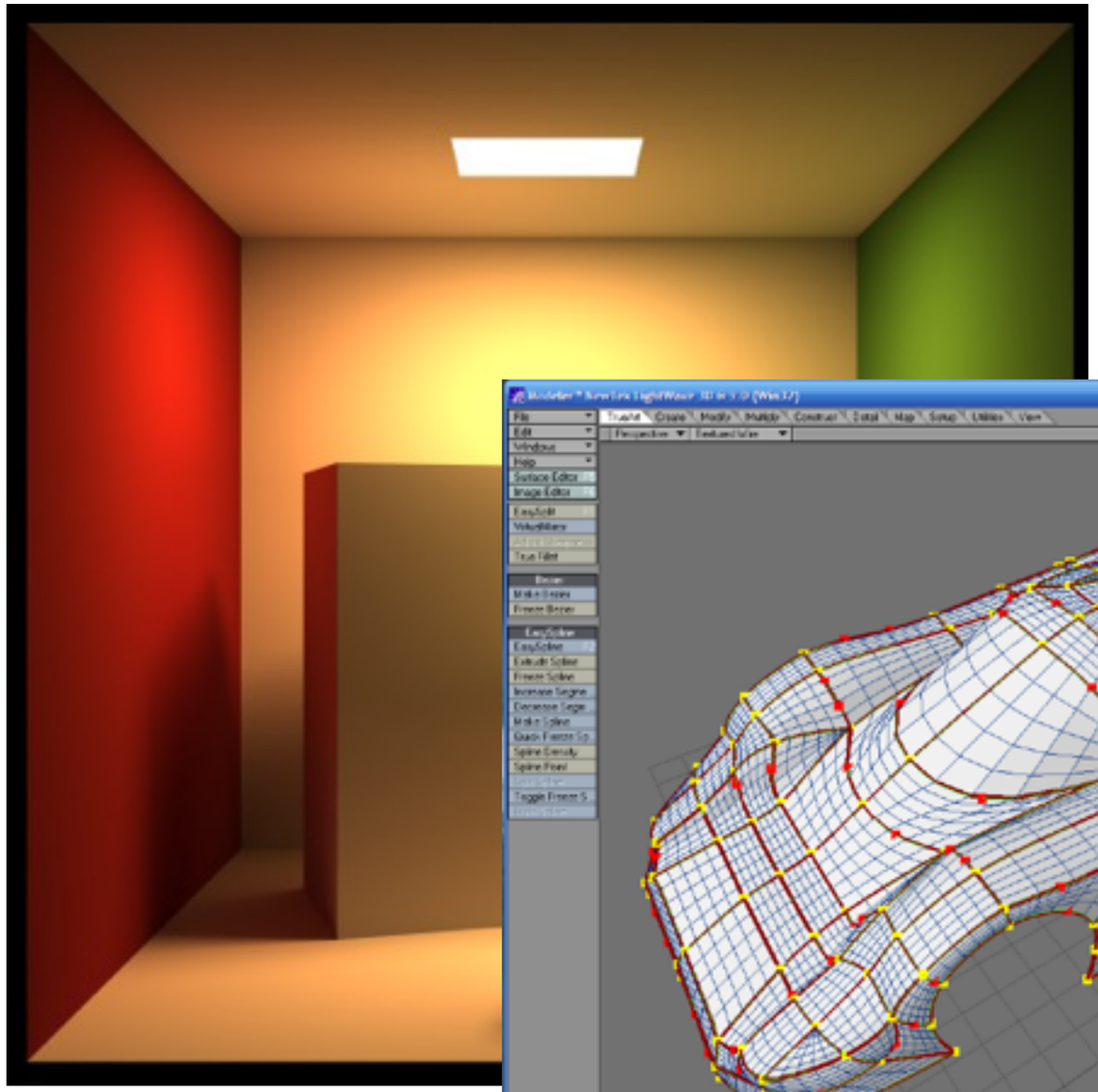
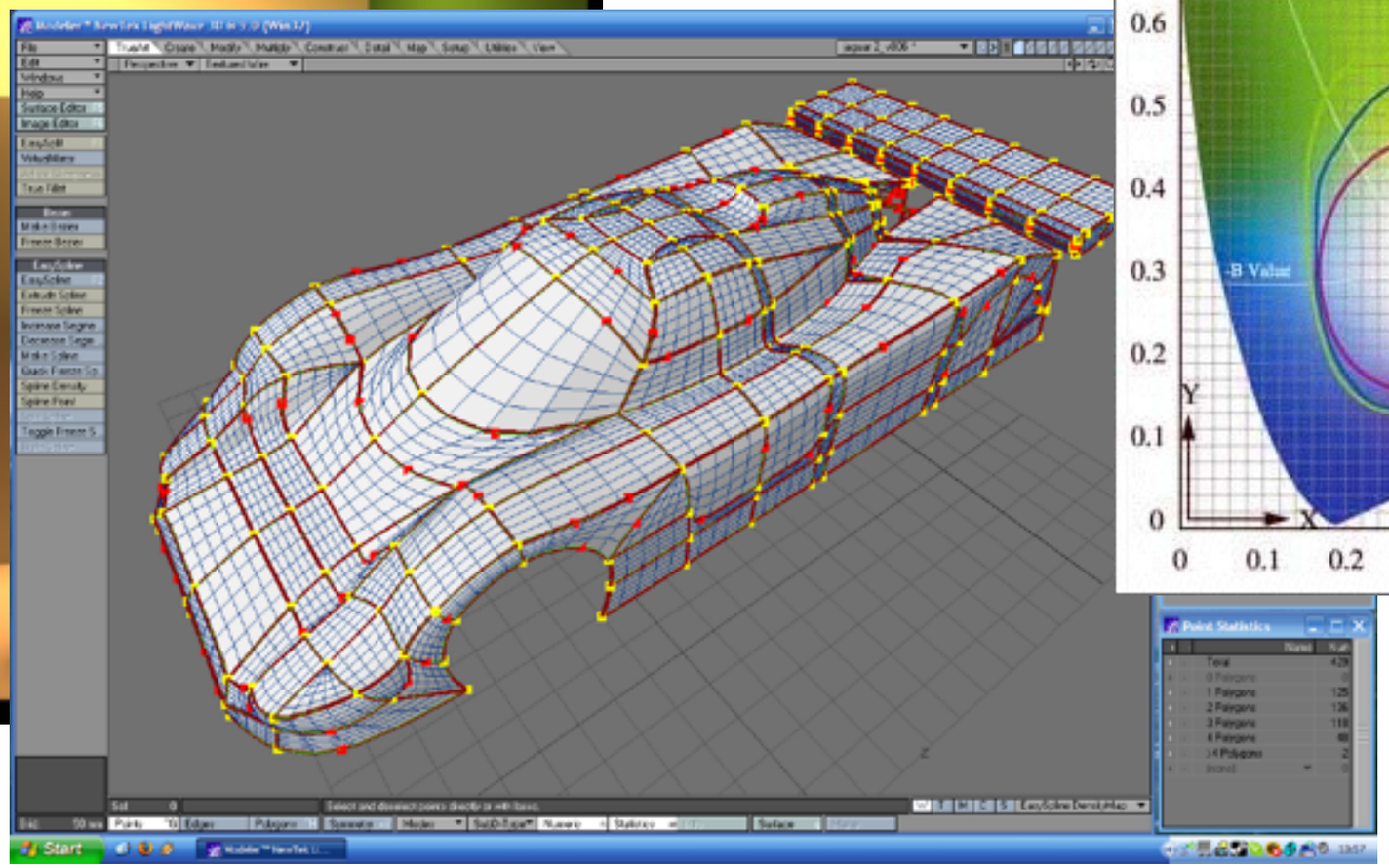
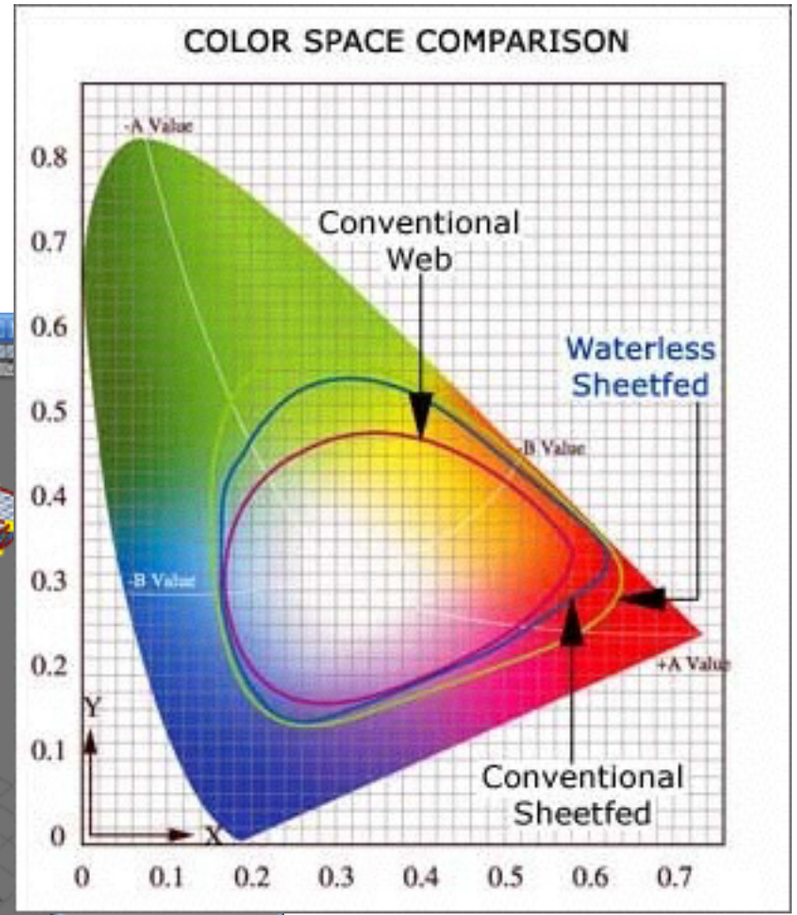






Lecture 17 : Introduction to Shading and the OpenGL Rendering Pipeline

Thursday November 4th 2021

Announcements

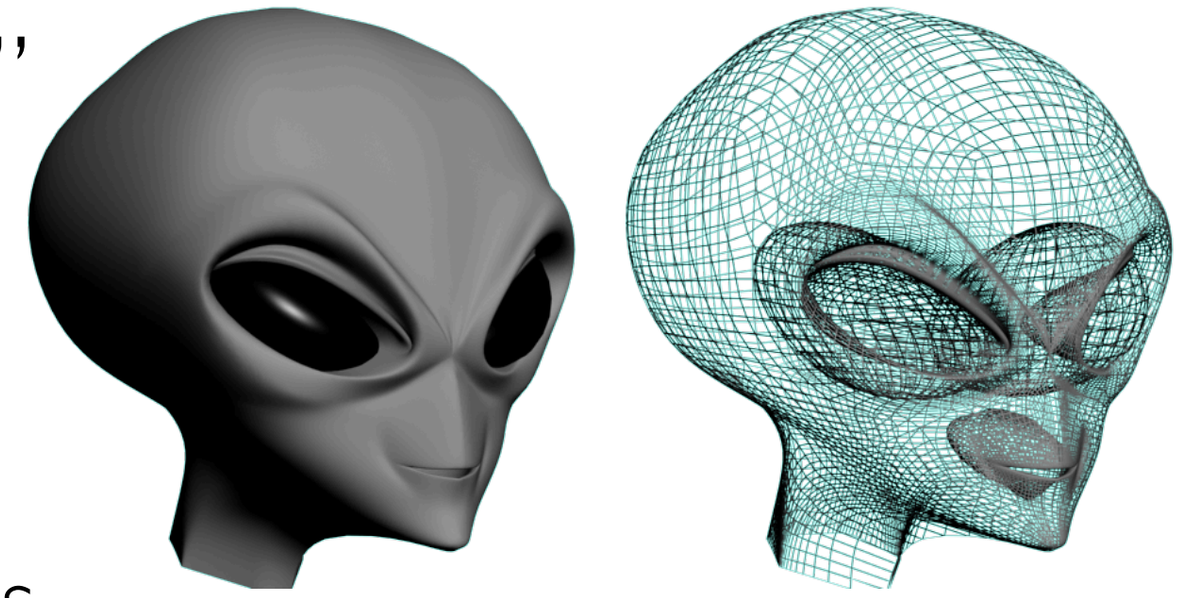
- Remember: Programming Assignment #4 has been posted, due Wednesday Nov 10th
- Today's office hours to be covered by TA Yutian Tao
- Midterm to be graded by next week

Today's lecture

- (Intuitive) introduction to shading & lighting
- The diffuse & lambertian models
- Introduction to the Graphics Rendering Pipeline

Lighting & Shading

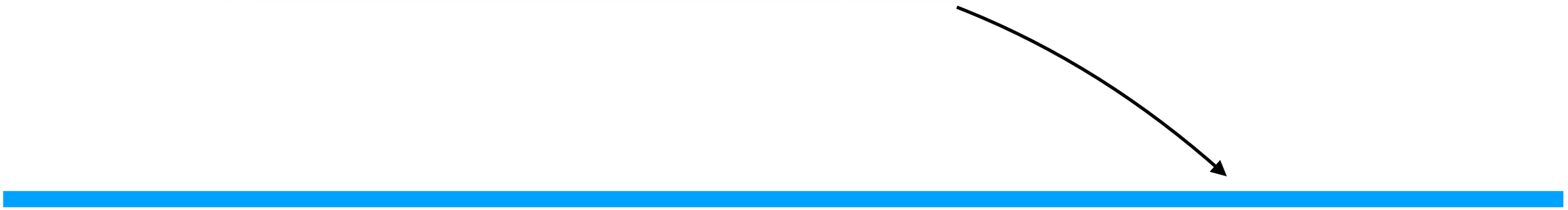
- So far we have focused on the *geometry* of 3D scenes, as far as drawing is concerned
  (i.e. what location on the screen or canvas will a specific point/line be drawn at)
- Shading/lighting tackles the question "what appearance will drawing primitives have?"
- Considerations
  - Color/shading
  - Interaction with light sources
  - Visibility, acceleration, etc



Lambertian reflectance

goo.gl/A81r7a

An object's surface being illuminated

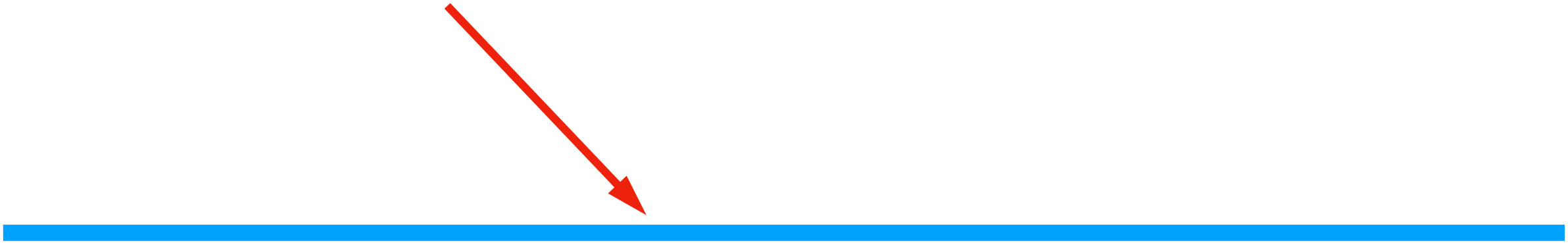


Lambertian reflectance

goo.gl/A81r7a

The Lambertian reflectance model:

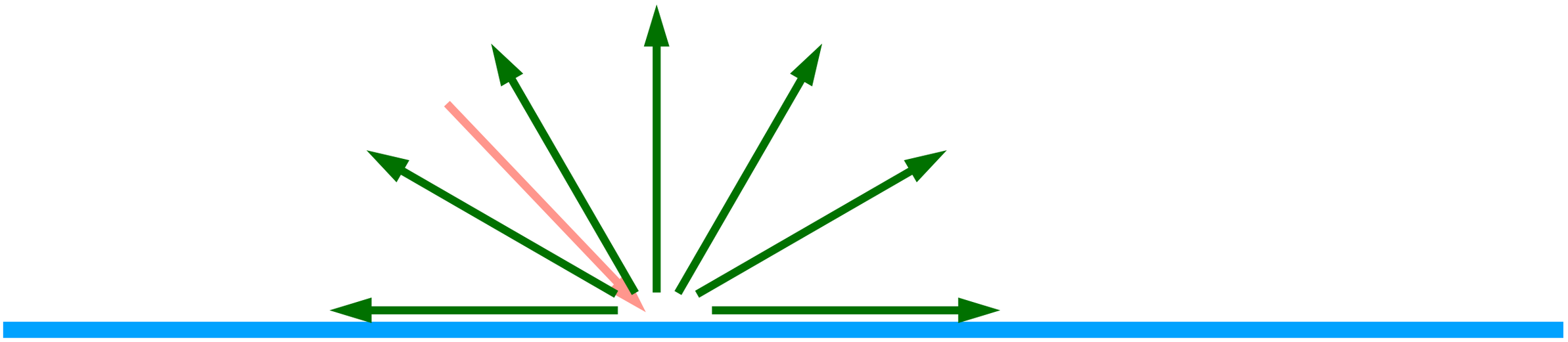
Any ray that falls on the surface …



Lambertian reflectance

goo.gl/A81r7a

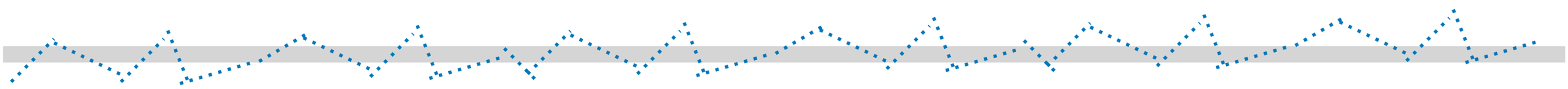
… has equal probability of being reflected in any possible direction



Lambertian reflectance

goo.gl/A81r7a

How could this be physically possible?
The “microfacet” model:
Surface is scattered with tiny, randomly oriented, reflective (mirror-like) facets



Lambertian reflectance

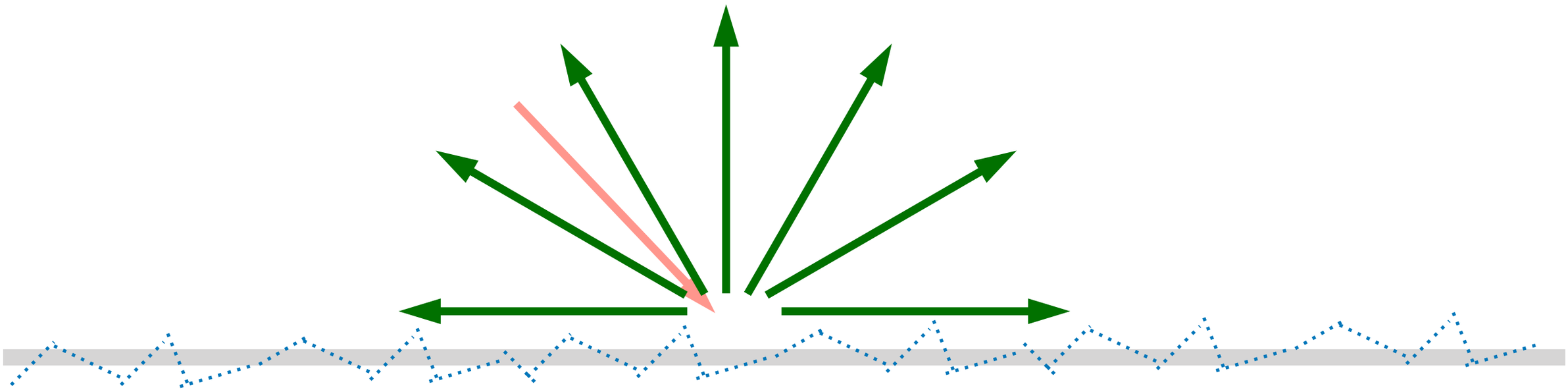
goo.gl/A81r7a

(also called a retro-reflective surface)

Lambertian reflectance

goo.gl/A81r7a

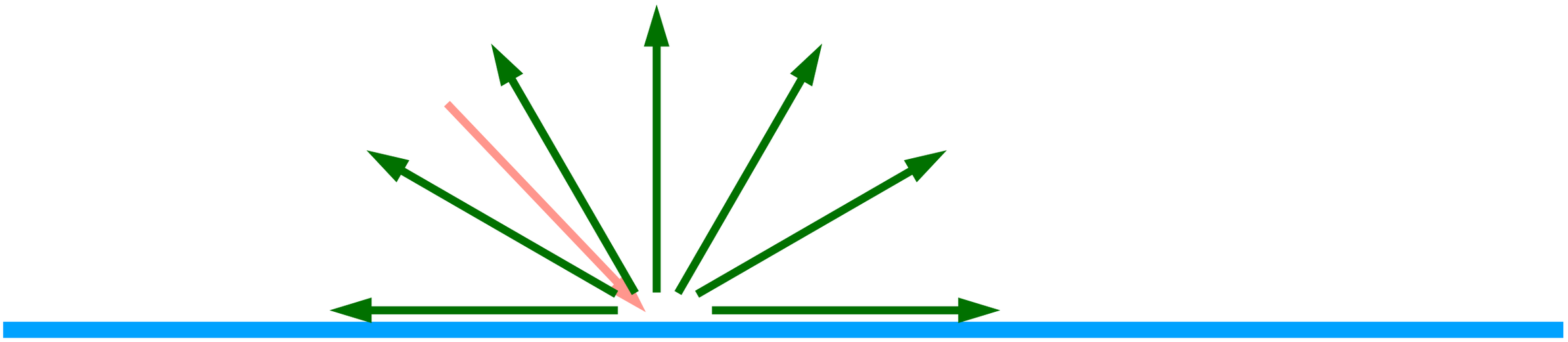
The property of uniformly-random direction of reflectance becomes a consequence of the fact that an incoming ray will reflect on a “randomly” oriented micro-mirror!



Diffuse reflection model

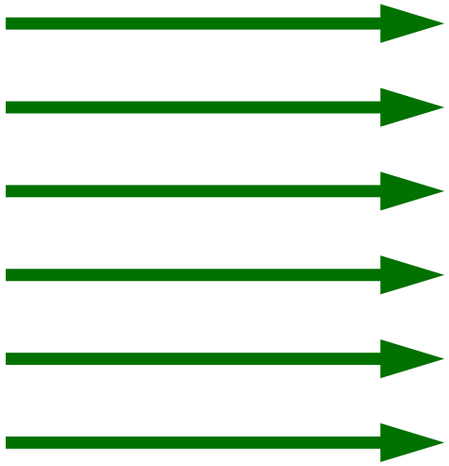
goo.gl/A81r7a

The appearance of an object modeled as a Lambertian surface is expressed by the *diffuse reflection model.*



Diffuse reflection model

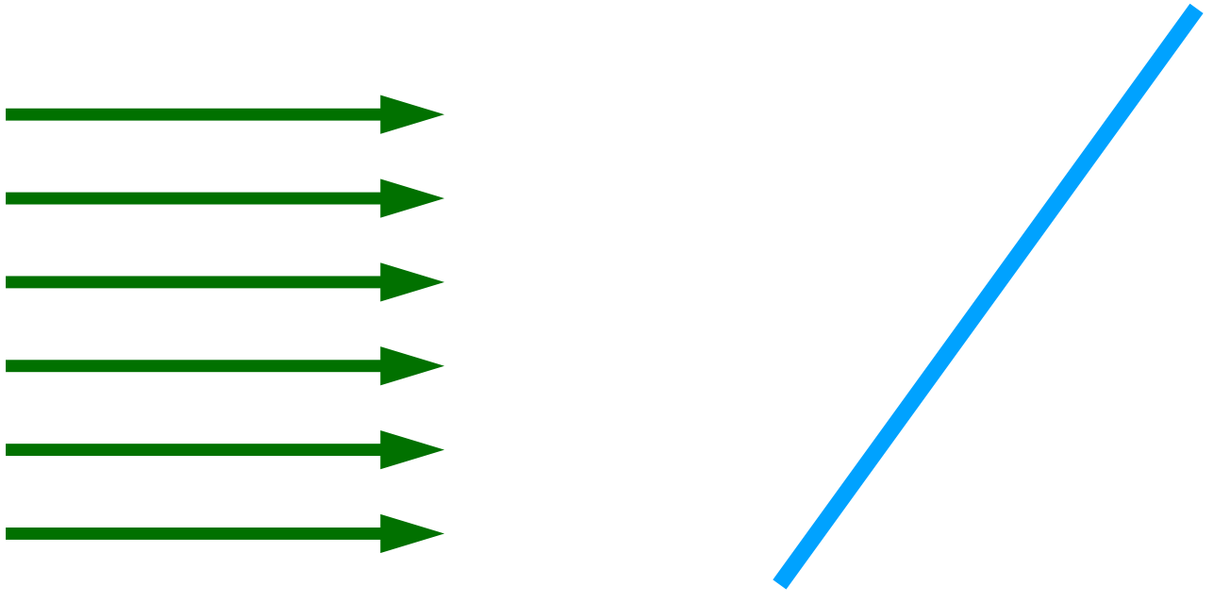
goo.gl/A81r7a



Consider a bundle of light rays
(“uniform” in the sense that the same
luminous energy falls on any given-area
surface that’s perpendicular to the bundle)

Diffuse reflection model

goo.gl/A81r7a

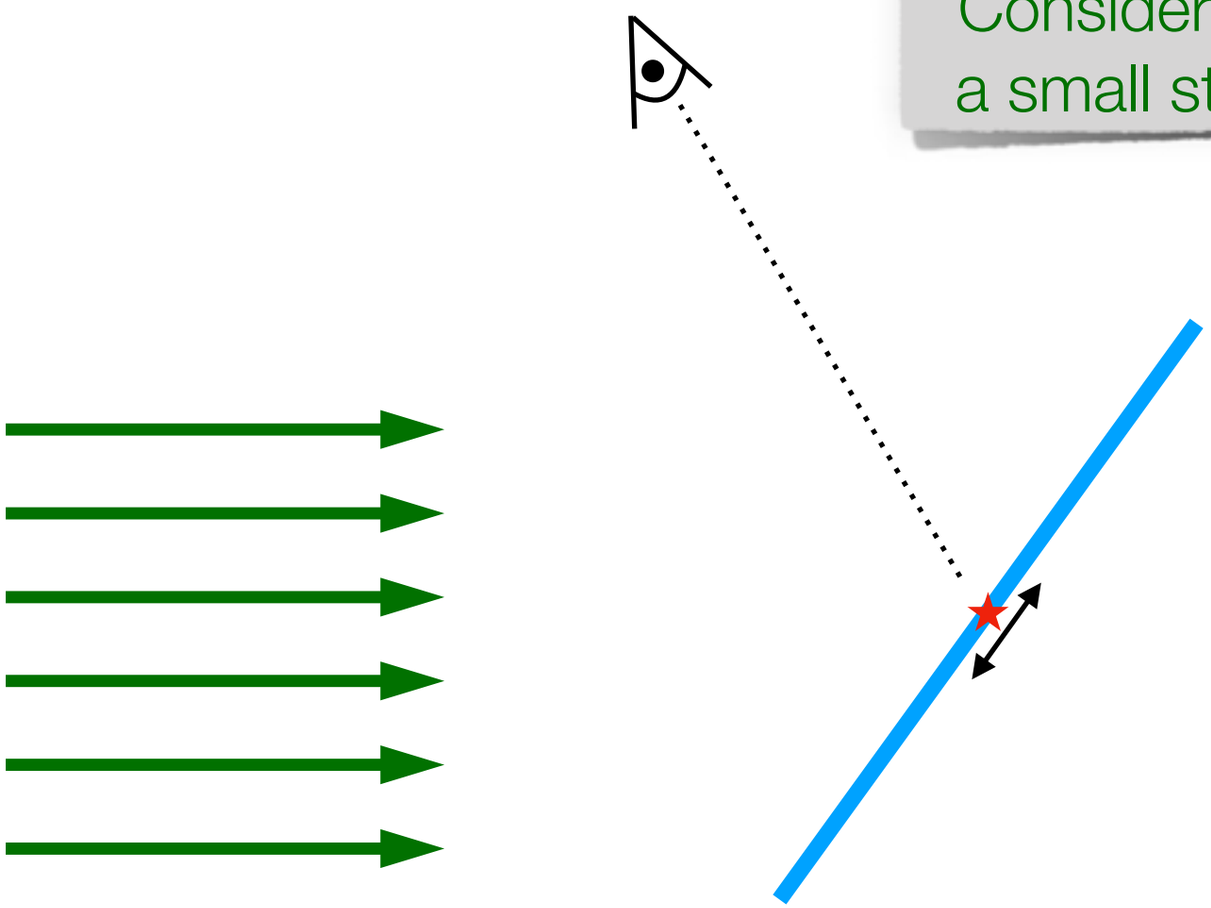


Also consider a surface that this light falls upon …

Diffuse reflection model

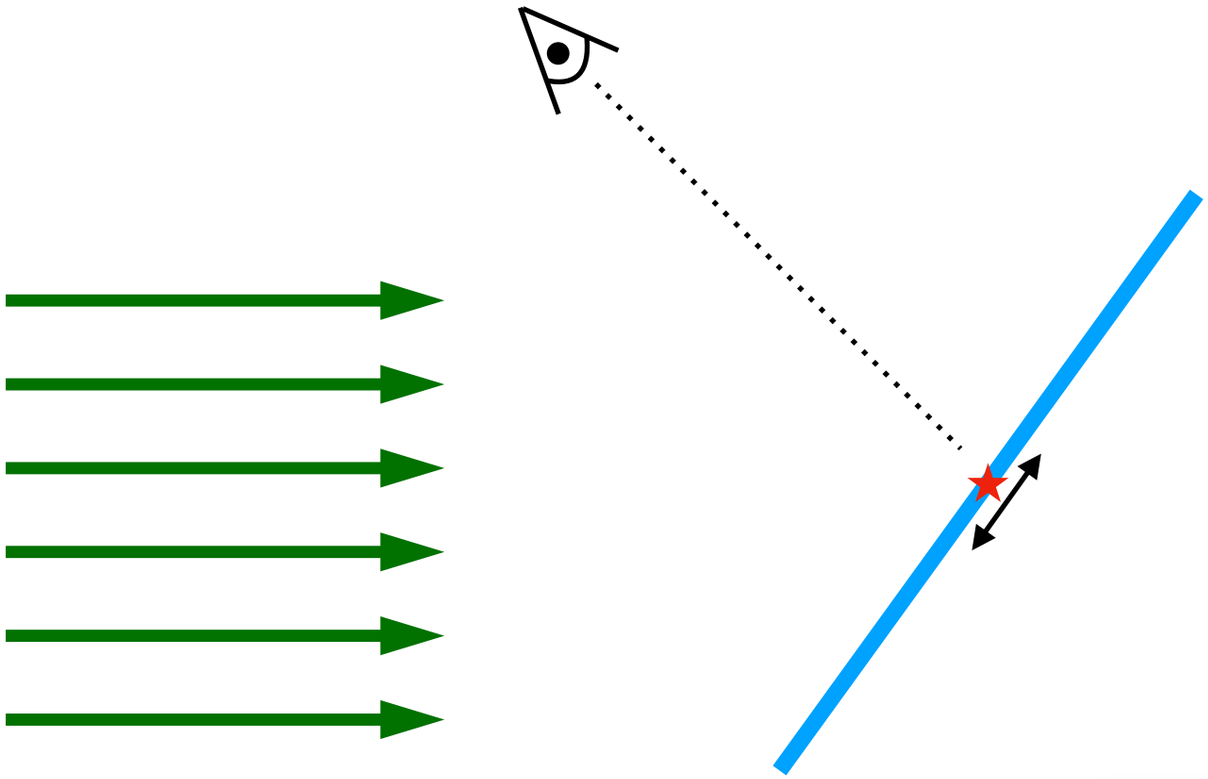
goo.gl/A81r7a

Consider an observer, looking at a small stretch of that surface …



Diffuse reflection model

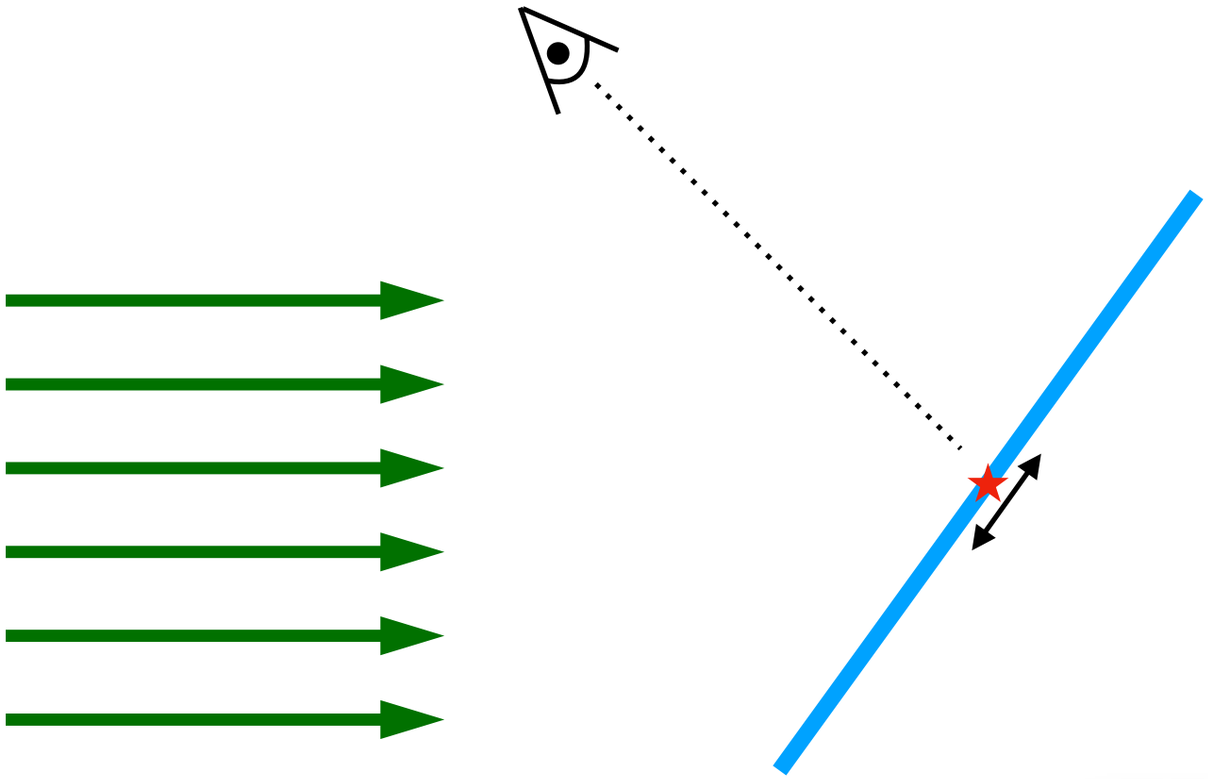
goo.gl/A81r7a



… due to the surface being Lambertian, the same amount of light gets reflected towards the observer, regardless of their placement

Diffuse reflection model

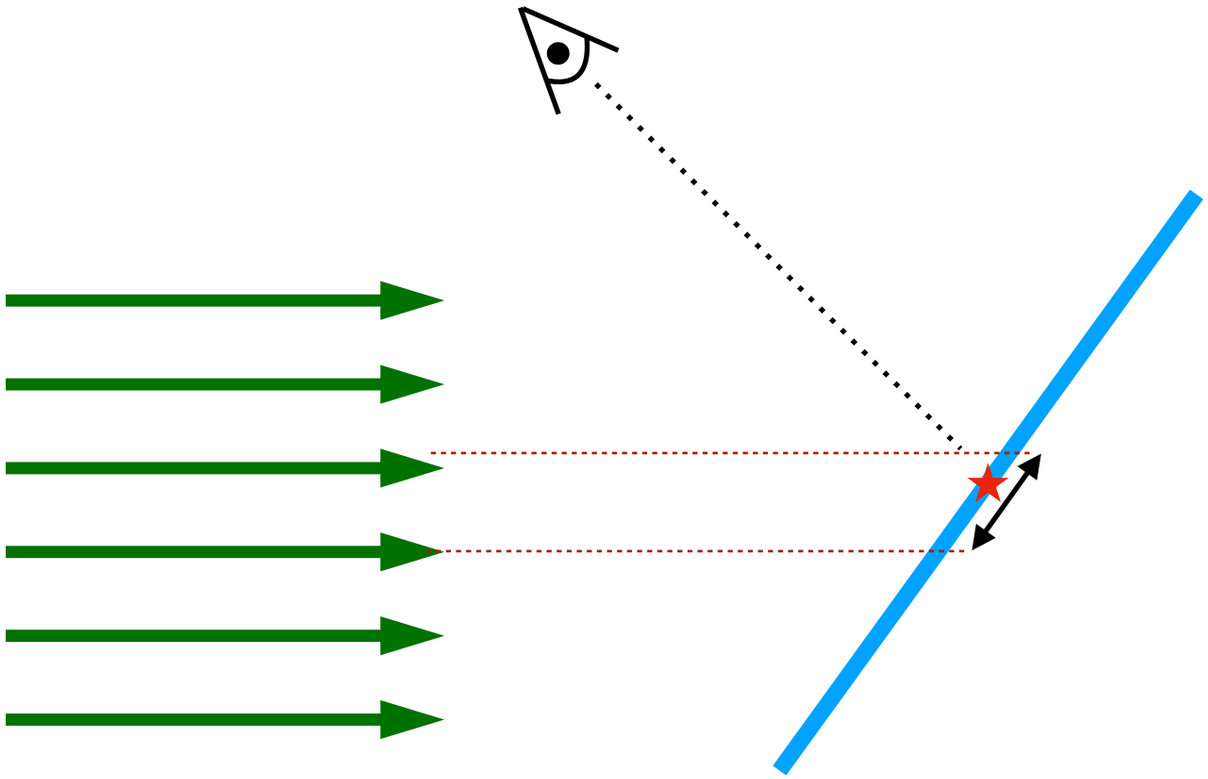
goo.gl/A81r7a



i.e. the same spot on the surface will appear as the same color, regardless of where we observe it from (as long as the light doesn't move)

Diffuse reflection model

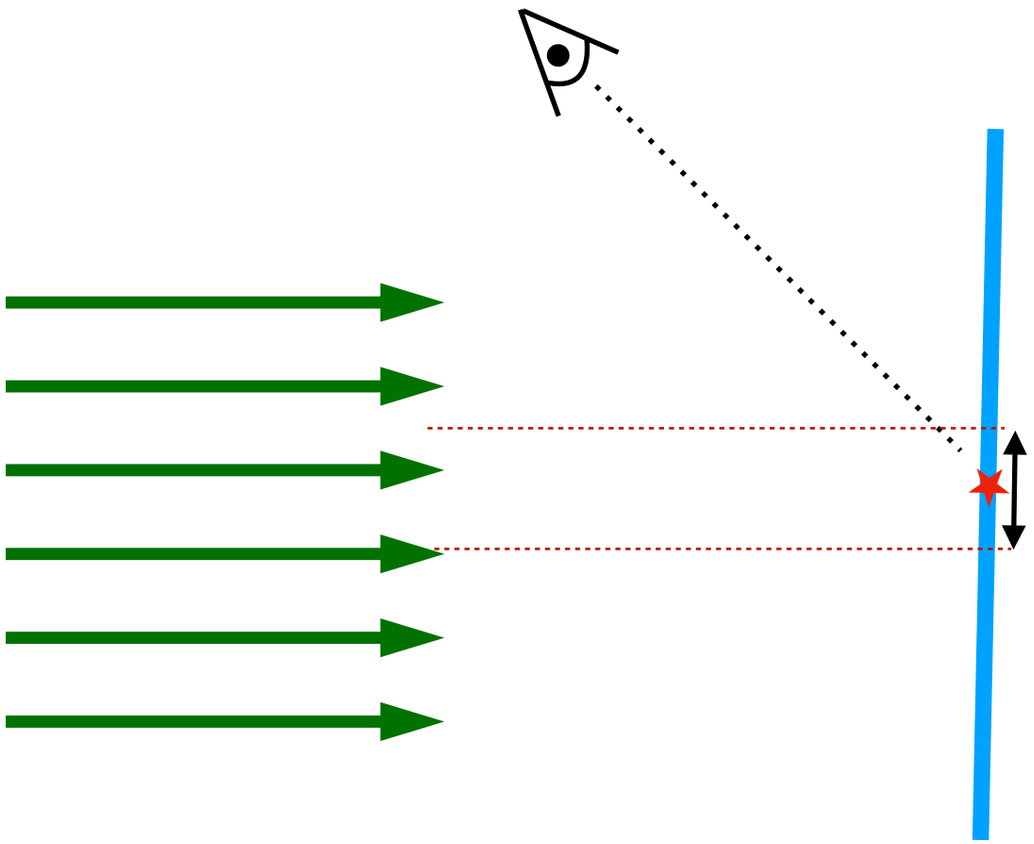
goo.gl/A81r7a



But, the amount of light that falls on this small stretch, depends on the orientation of the surface!

Diffuse reflection model

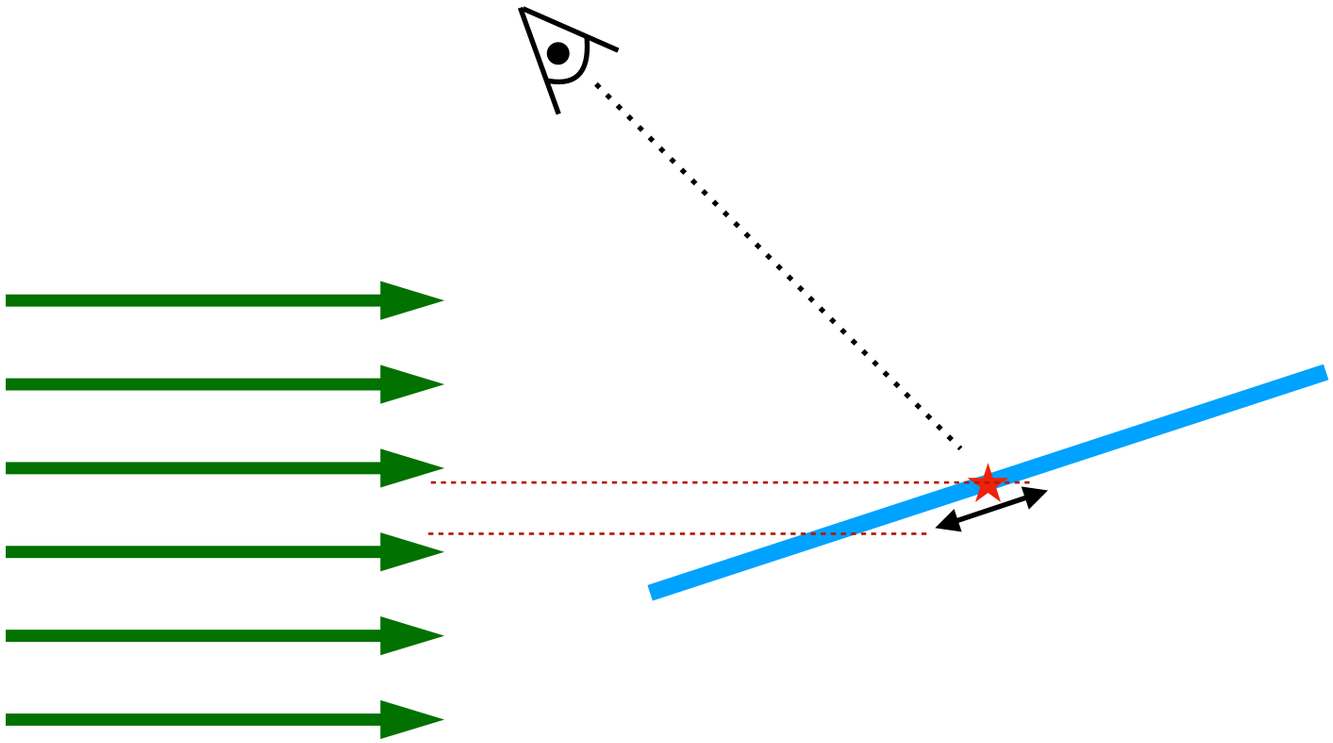
goo.gl/A81r7a



Surfaces that are more "perpendicular" to the light collect more incident light energy

Diffuse reflection model

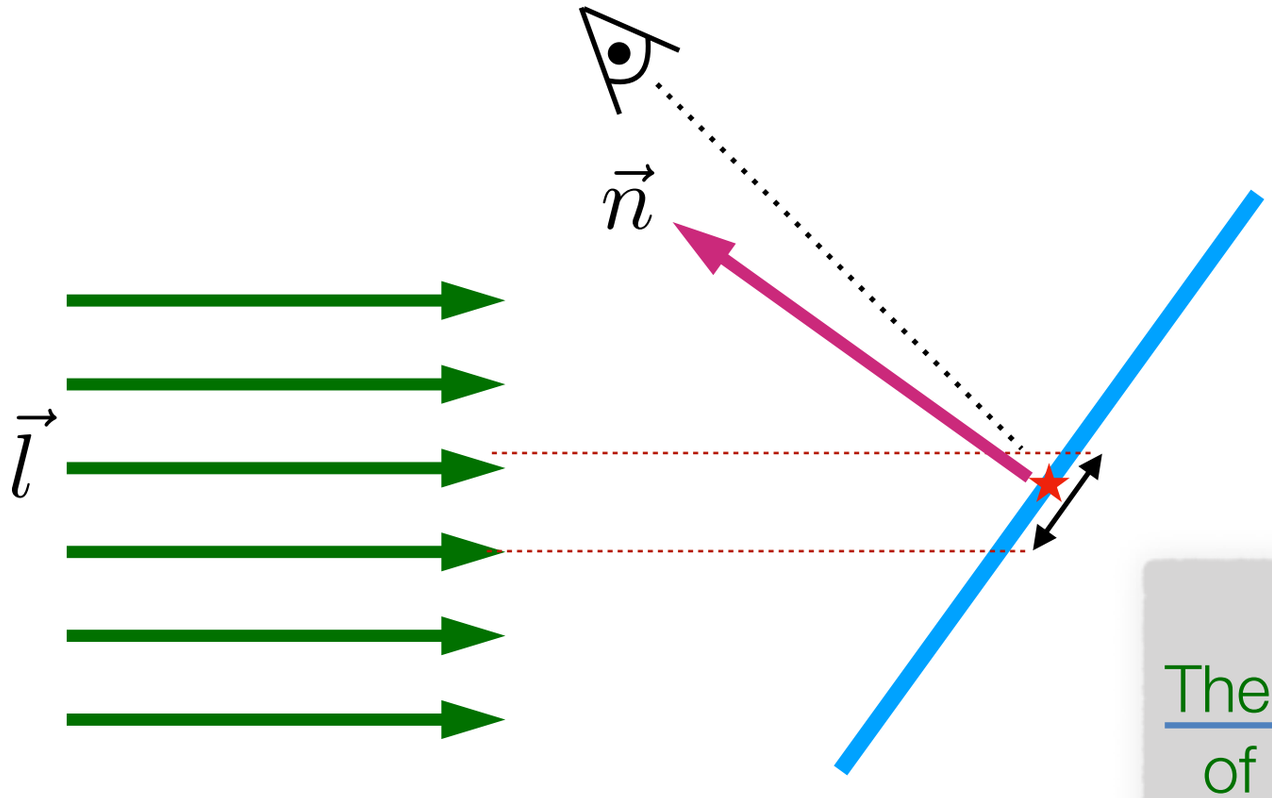
goo.gl/A81r7a



… and more “slanted” surfaces receive less!

Diffuse reflection model

goo.gl/A81r7a



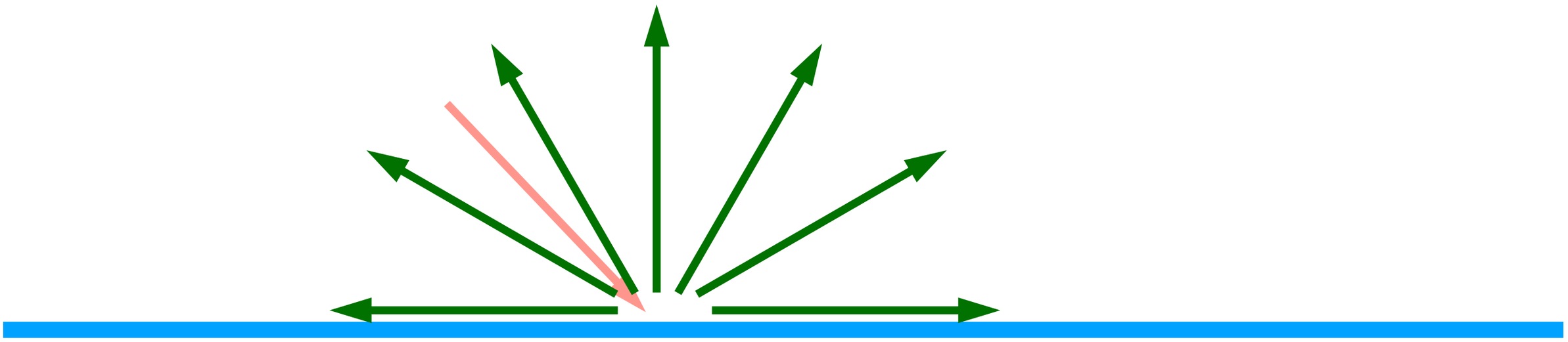
The diffuse reflection model:
The light incident on any small stretch of the surface is proportional to the cosine between the light direction and the surface normal!

$$I_d = k_d(\vec{l} \cdot \vec{n})$$

Specular reflection model

goo.gl/ooE6NL

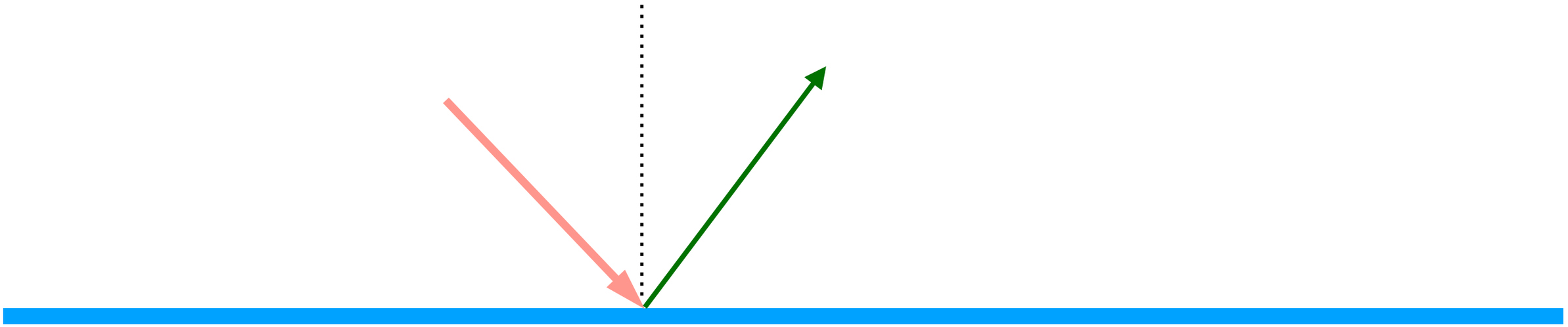
Diffuse reflection intuitively pairs with "matte" objects



Specular reflection model

goo.gl/ooE6NL

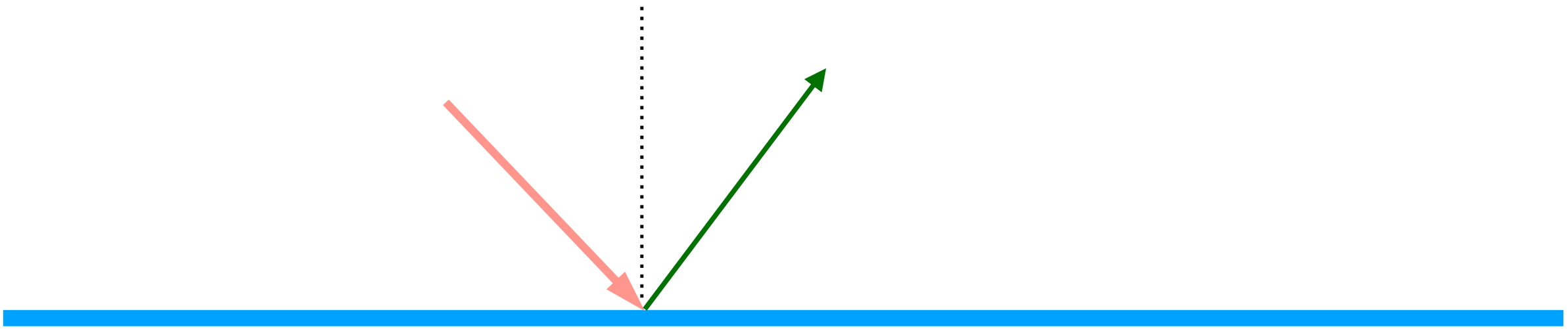
But polished, “mirror-like” objects typically reflect all light in the symmetric direction …



Specular reflection model

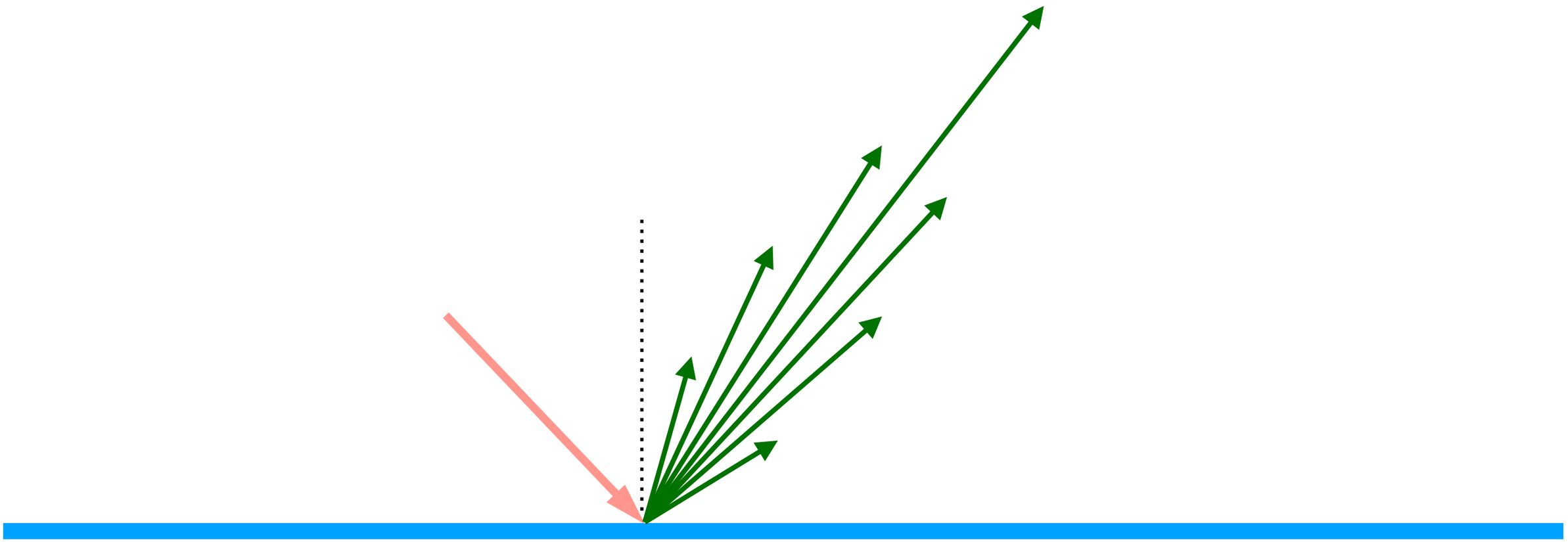
goo.gl/ooE6NL

… then again, there are shiny objects that are not perfect mirrors (even mirrors themselves are not "perfect", in fact!)



Specular reflection model

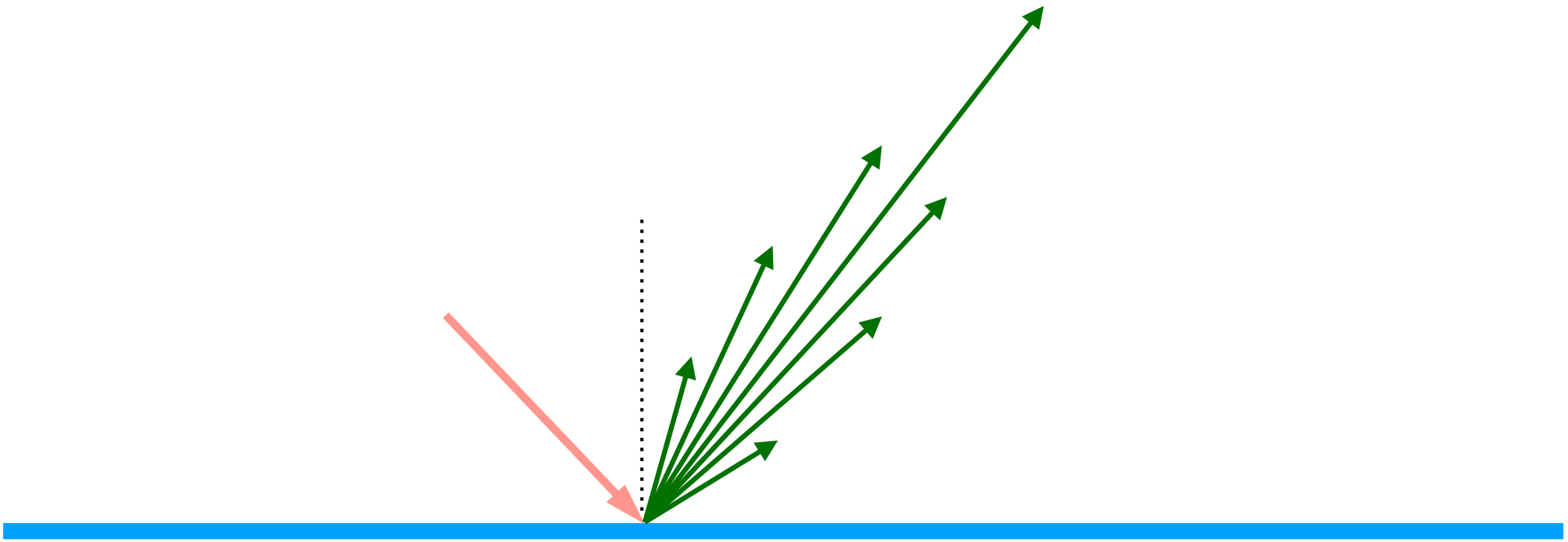
goo.gl/ooE6NL



More realistically, the perfect-reflection direction is merely the “statistically preferred” direction …

Specular reflection model

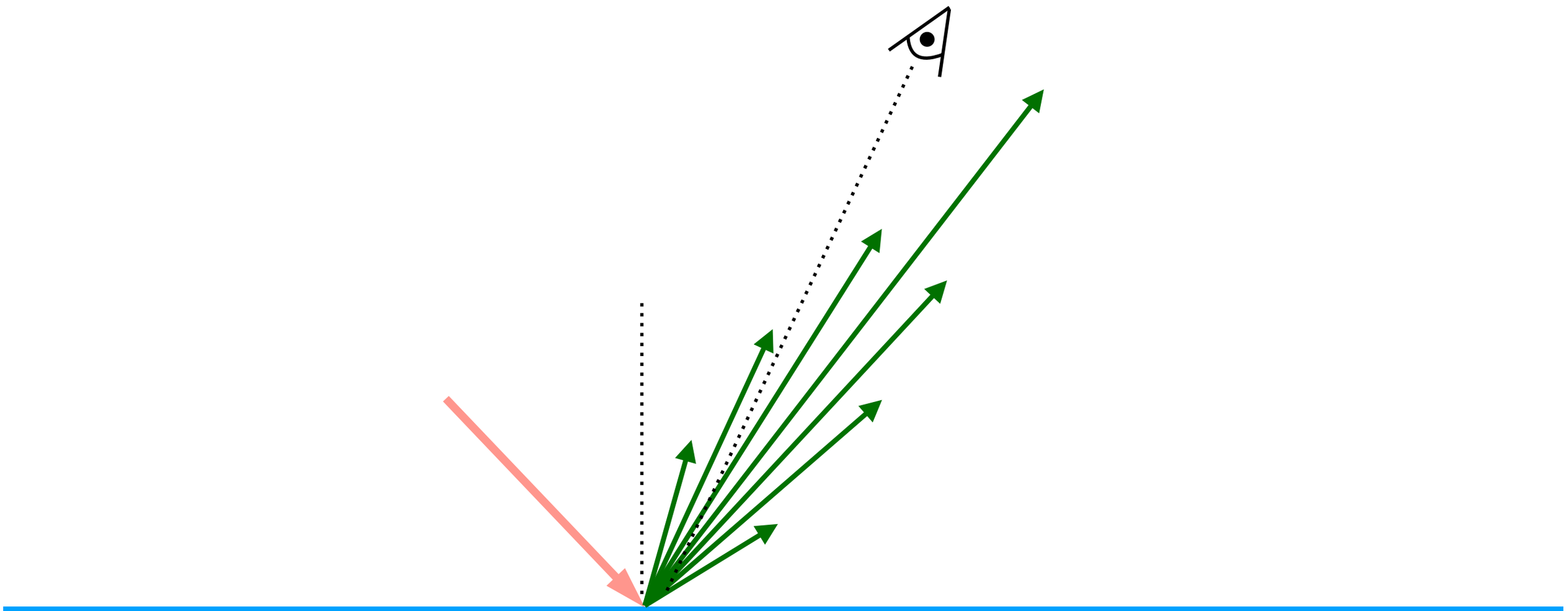
goo.gl/ooE6NL



… but nearby directions are also likely to have *some fraction* of light reflected towards them.

Specular reflection model

goo.gl/ooE6NL



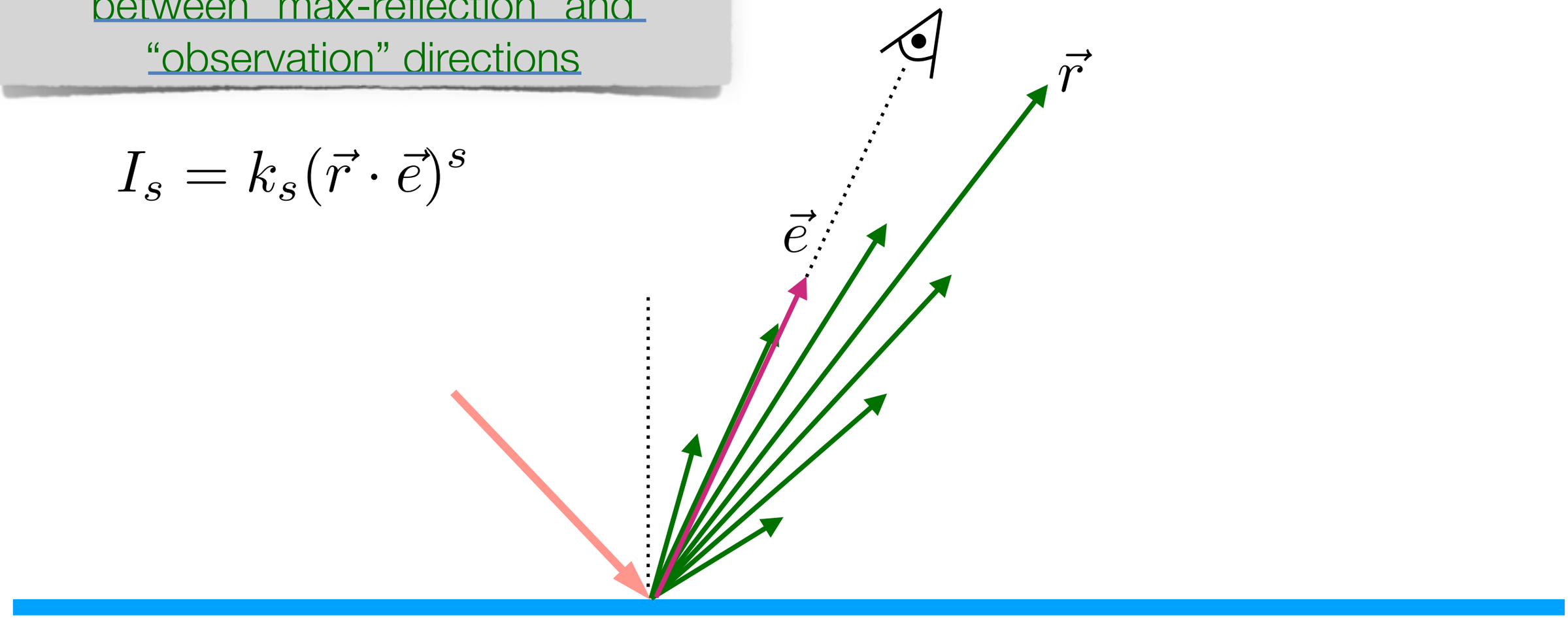
Ultimately, the question becomes:
How close is any given direction of observation to the direction of "maximum" intensity reflection?

Specular reflection model

goo.gl/ooE6NL

Typical “specular reflection model”:
Reflected light intensity is proportional to *some power* of the dot product between “max-reflection” and “observation” directions

$$I_s = k_s(\vec{r} \cdot \vec{e})^s$$



Phong reflection model

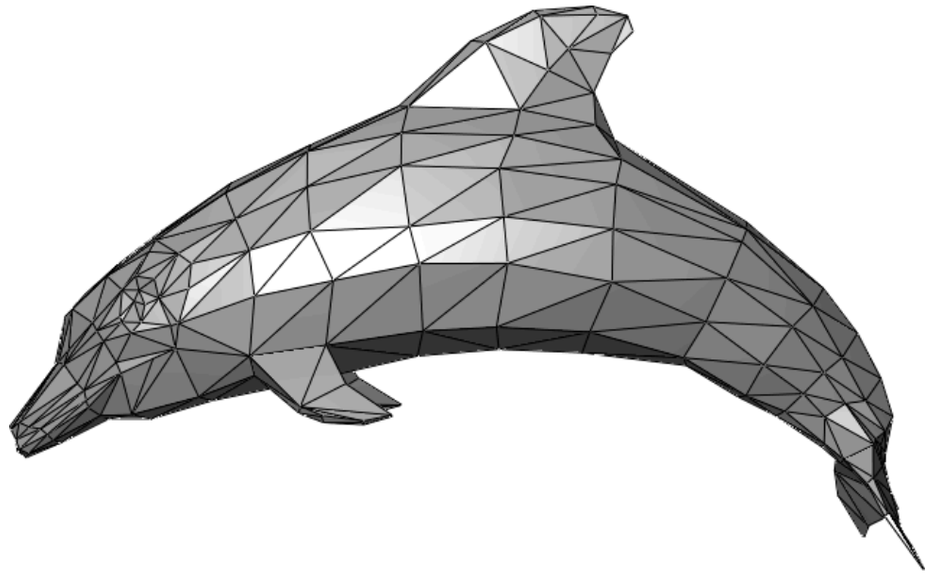
goo.gl/0pO3lK

Mixture of diffuse, specular and *ambient light/shading*

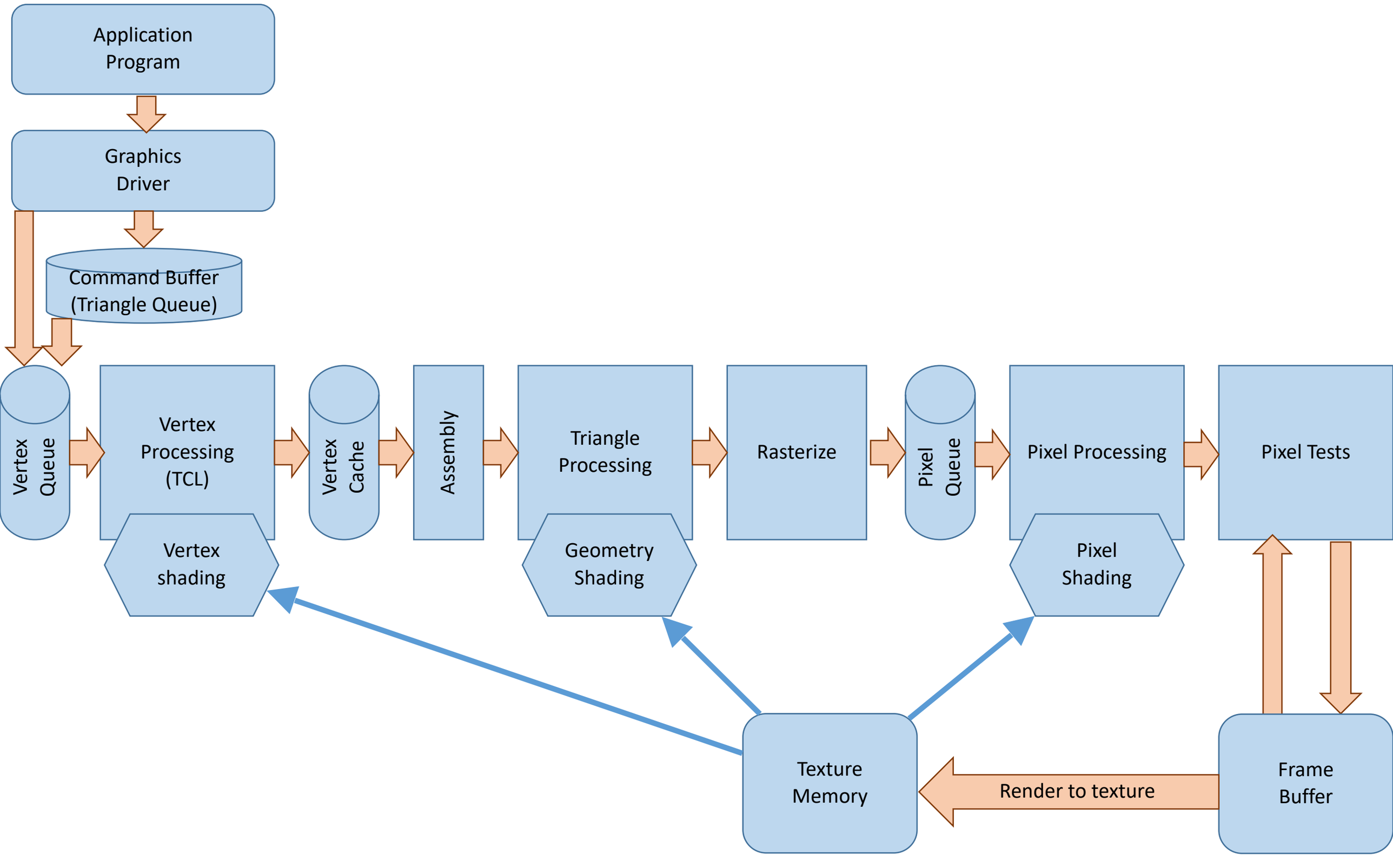
Ambient lighting is constant *regardless* of the relative placement of eye/light/object!

Graphics primitives?

- In order to have "regions" of a 3D model that color(s) are assigned to, we need to split up our models into surface elements.
- Triangles are the workhorse of the GPU rendering pipeline.
- Good approximations to other shapes, simple enough to make really fast.



The (GPU) graphics pipeline



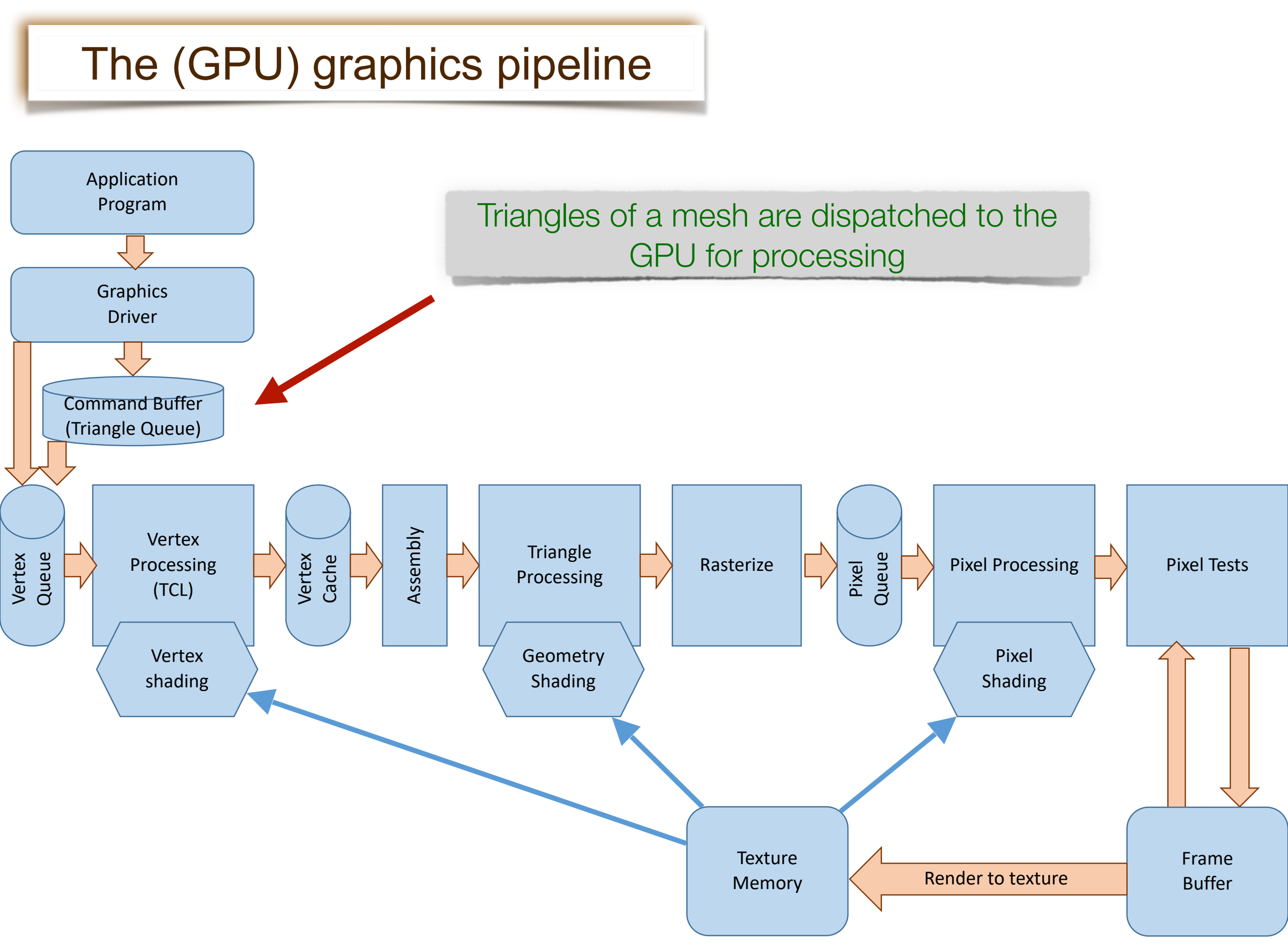
The (GPU) graphics pipeline

Application Program

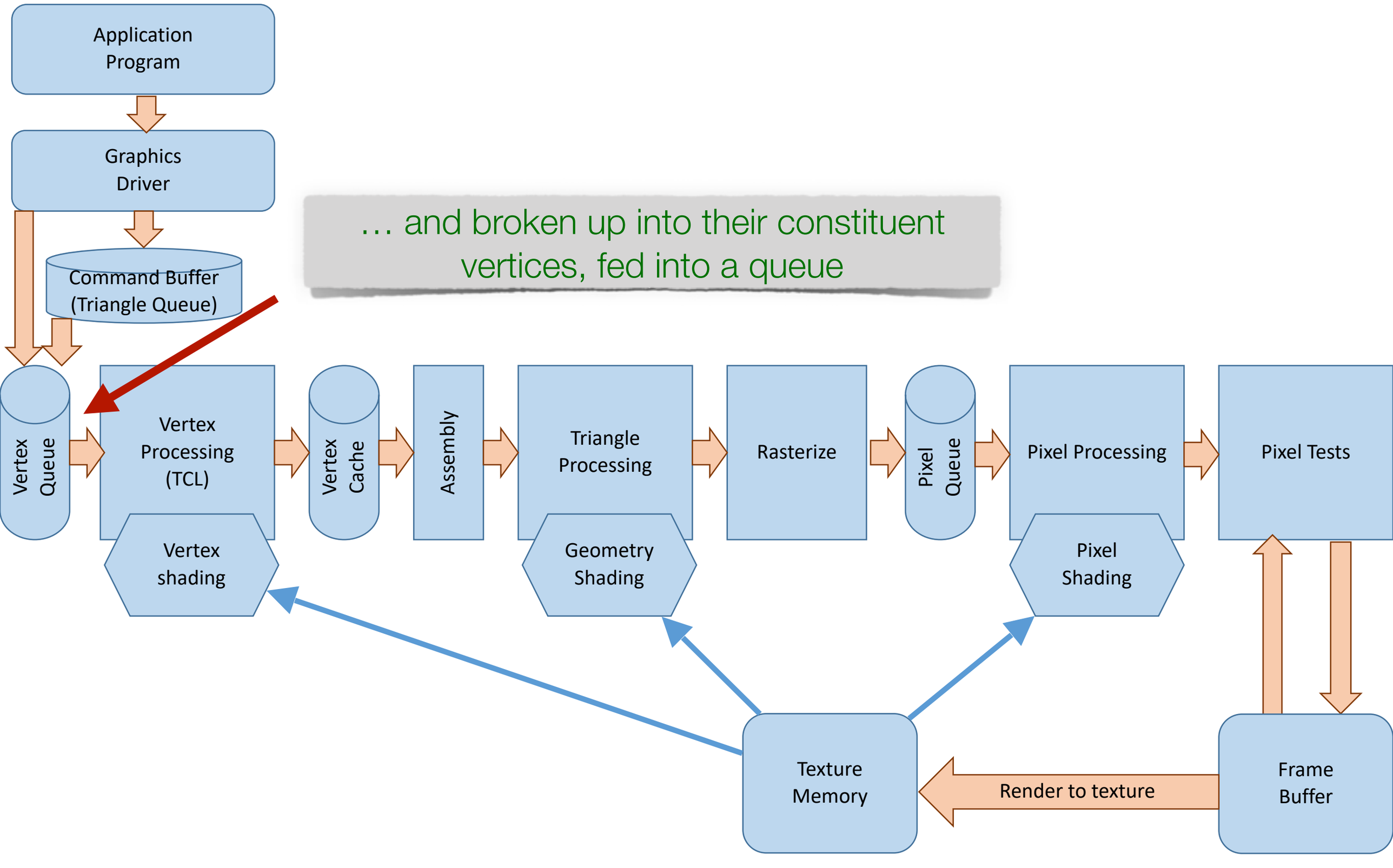
Graphics Driver

Command Buffer (Triangle Queue)

Triangles of a mesh are dispatched to the GPU for processing

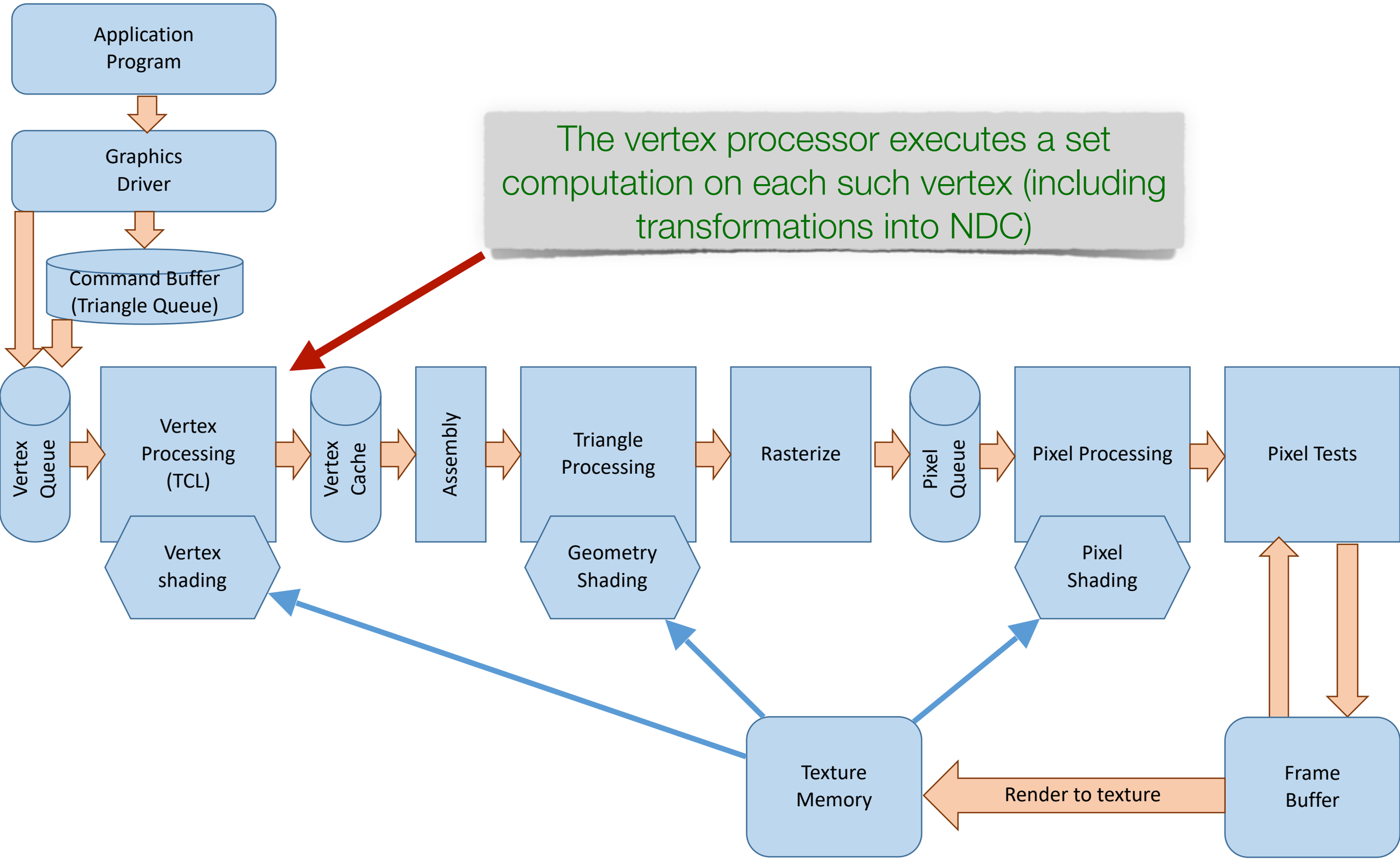


The (GPU) graphics pipeline

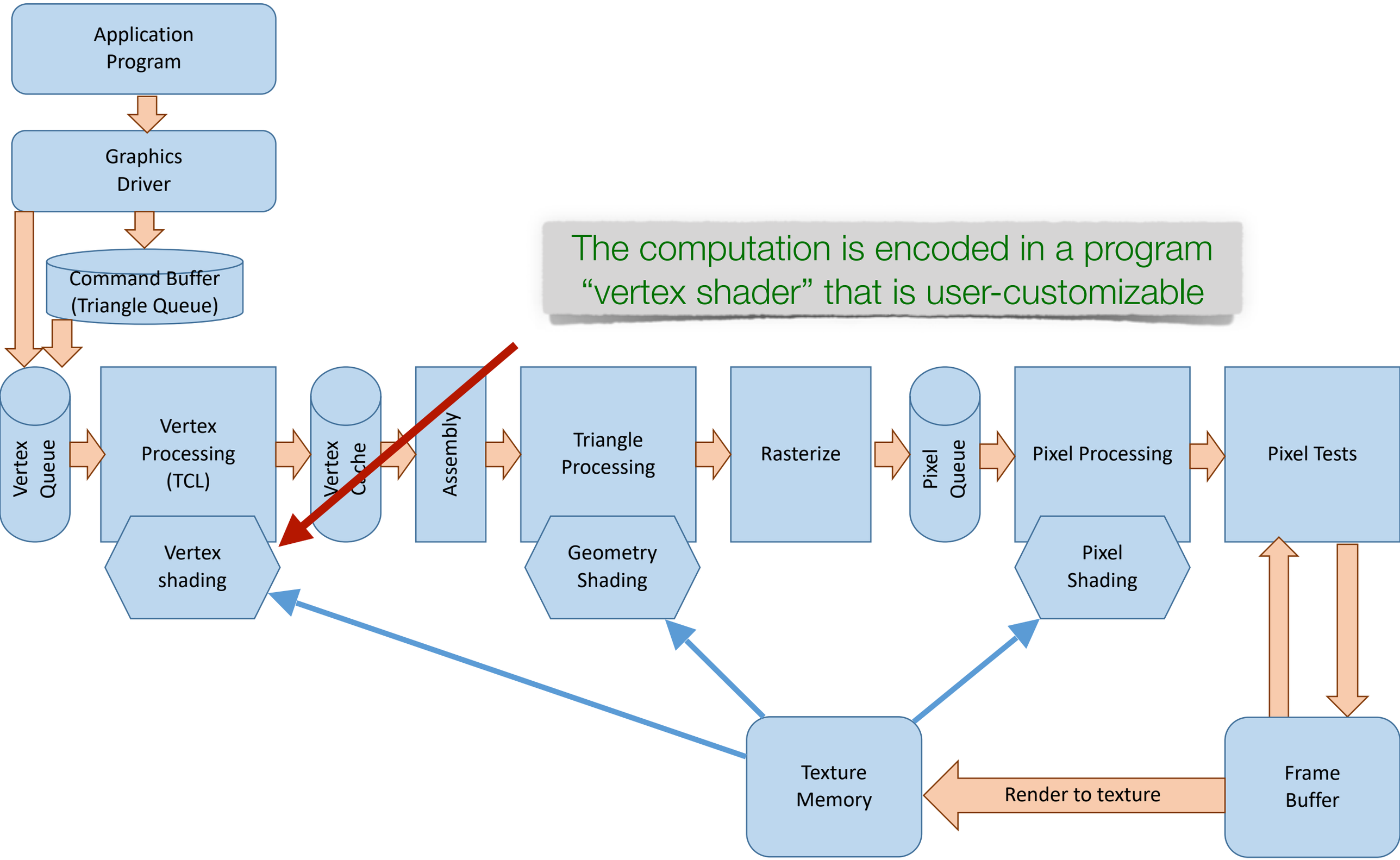


… and broken up into their constituent vertices, fed into a queue

The (GPU) graphics pipeline



The (GPU) graphics pipeline



The computation is encoded in a program "vertex shader" that is user-customizable

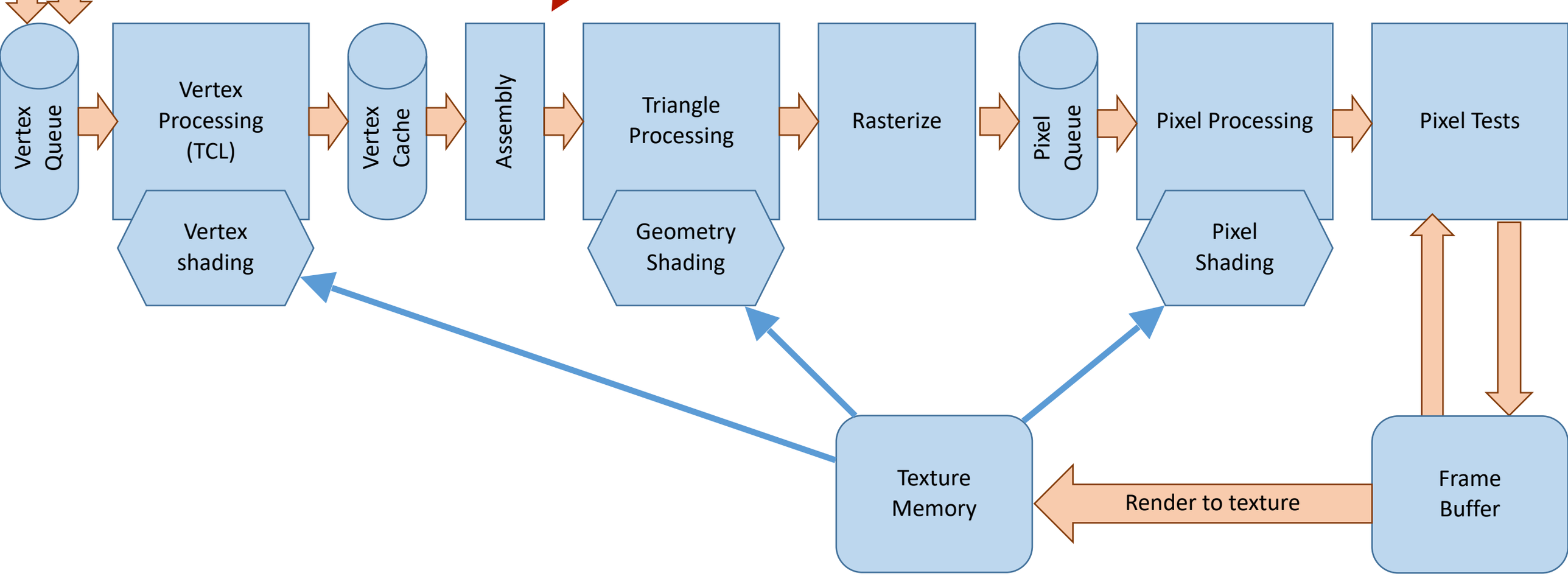
The (GPU) graphics pipeline

Application Program

Graphics Driver

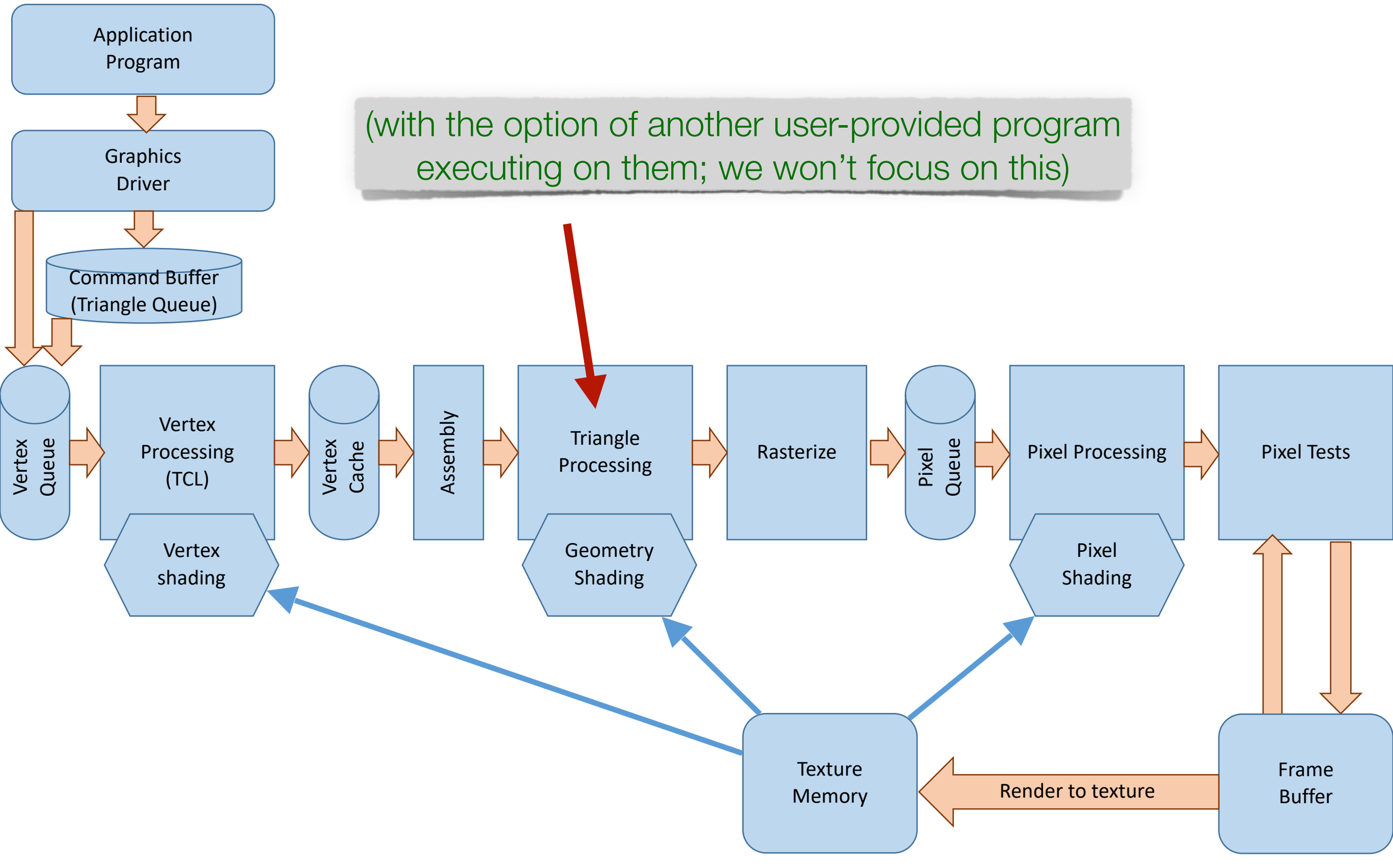
Command Buffer (Triangle Queue)

Transformed vertices are again assembled into triangles



The (GPU) graphics pipeline

(with the option of another user-provided program executing on them; we won't focus on this)



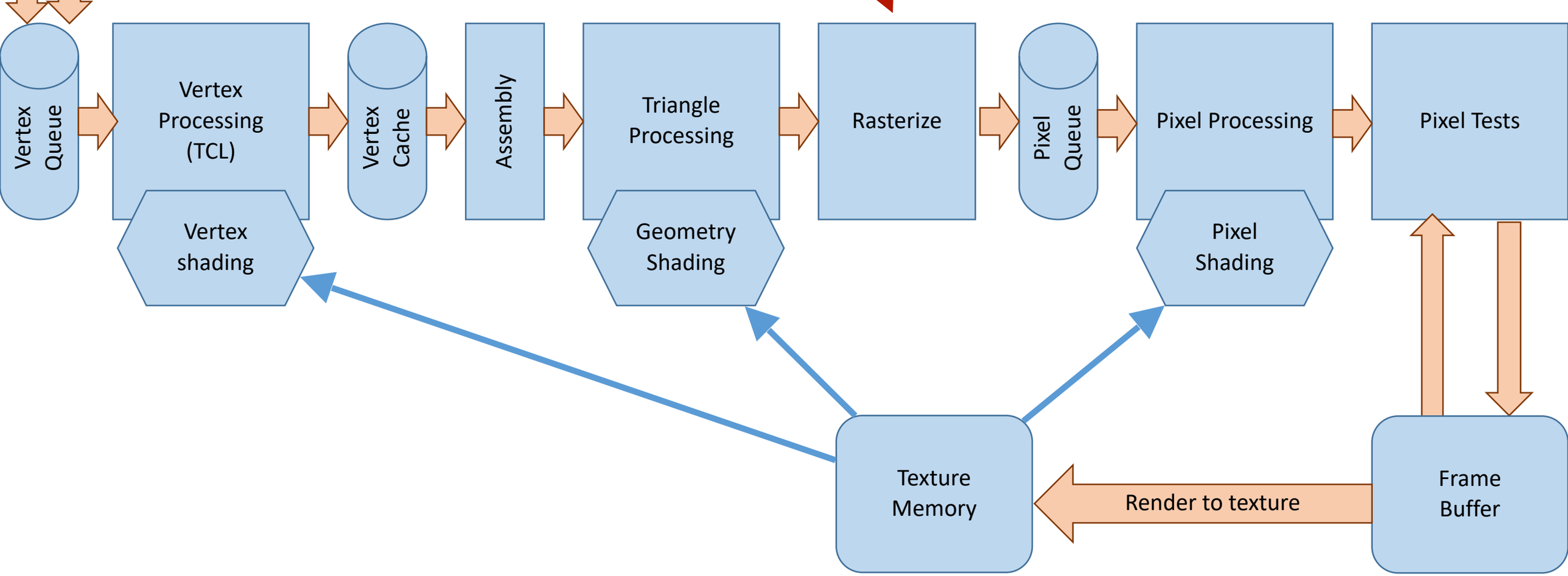
The (GPU) graphics pipeline

Application Program

Graphics Driver

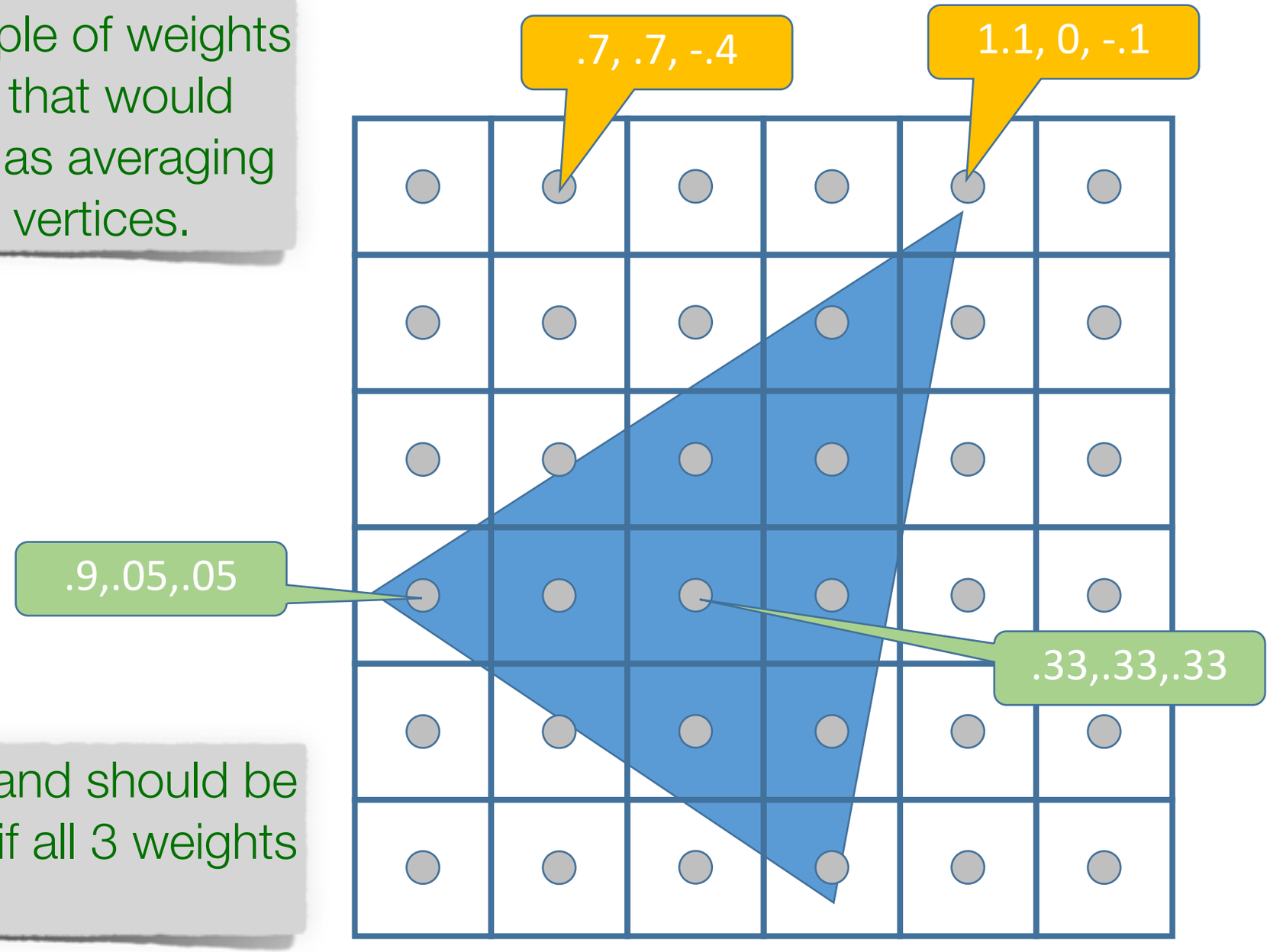
Command Buffer (Triangle Queue)

Each triangle is then converted into a collection of pixels that cover its spatial extent



Rasterization (in hardware)

For every pixel, compute a triple of weights (“barycentric coordinates”) that would reconstruct the point if used as averaging weights from the triangle vertices.



A pixel is *inside the triangle* (and should be “rasterized” into a fragment) if all 3 weights are positive.

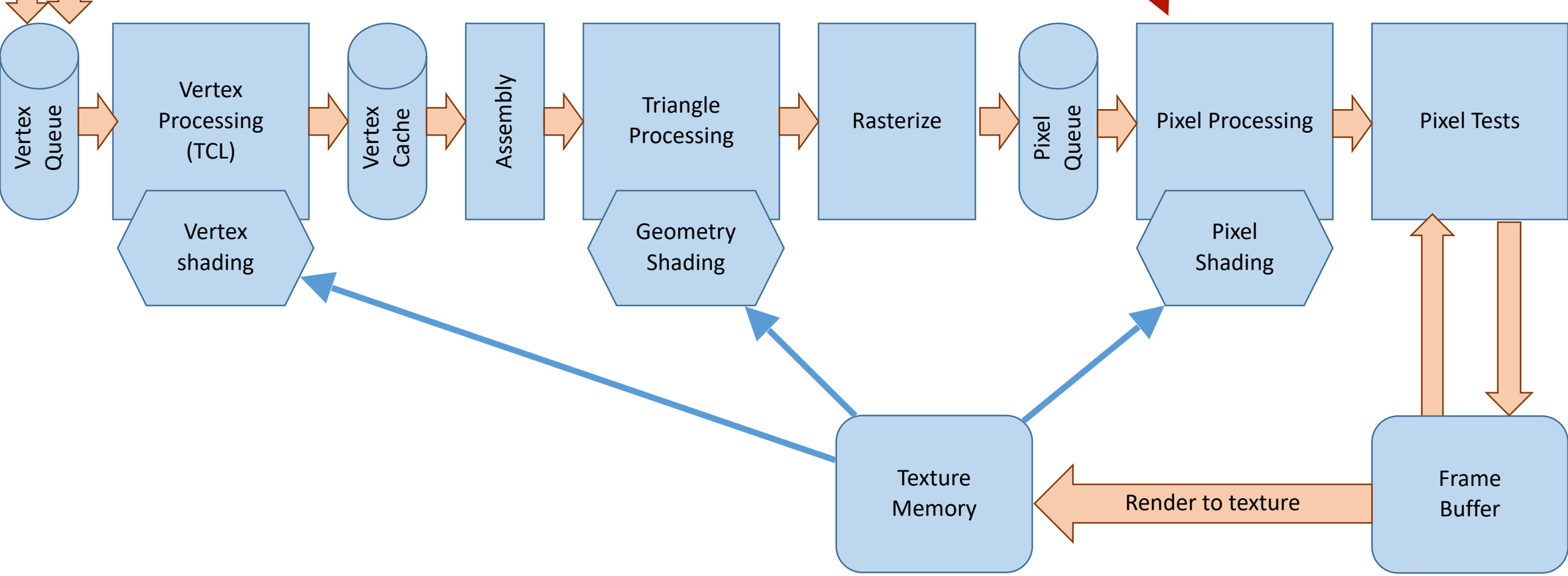
The (GPU) graphics pipeline

Application Program

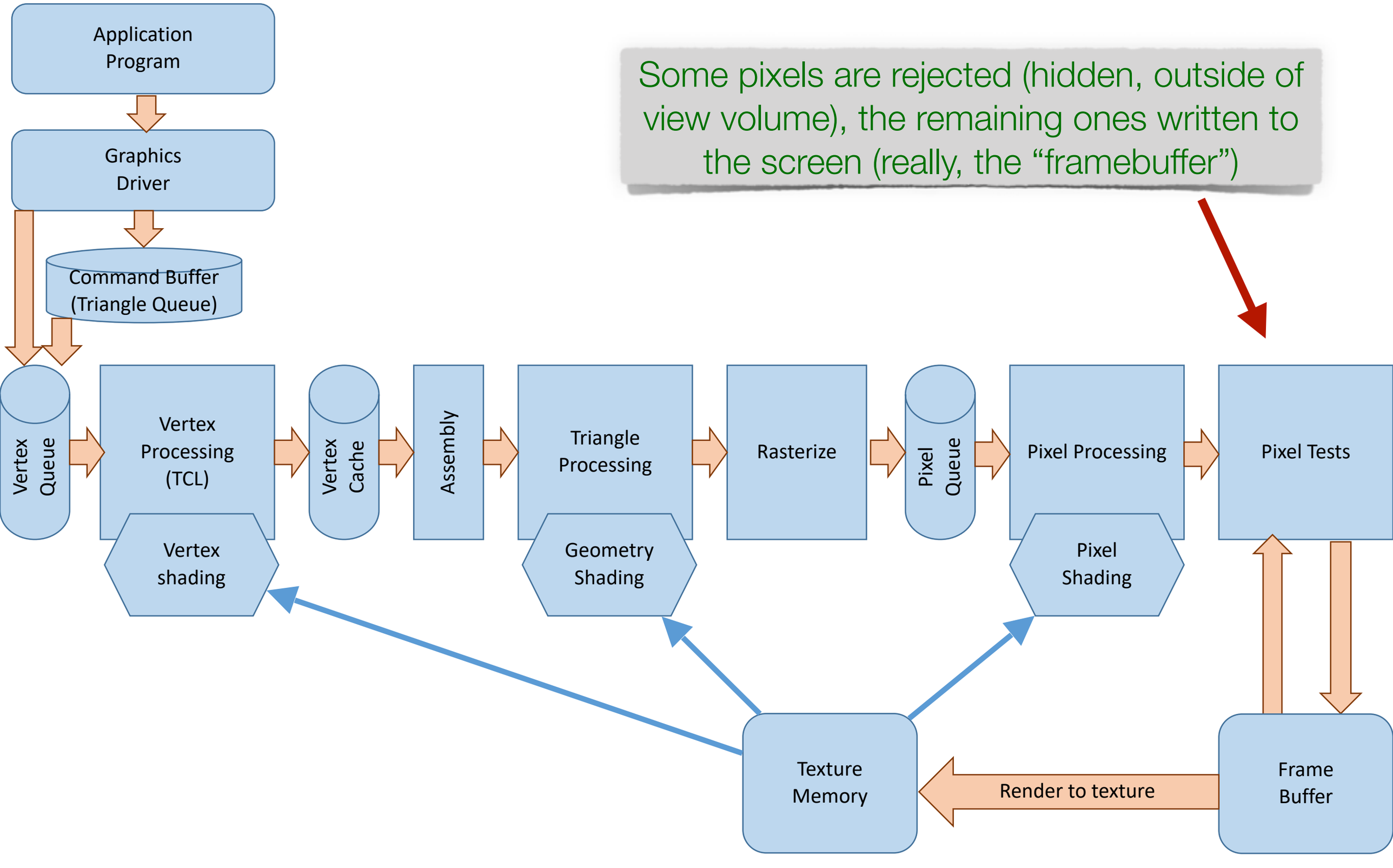
Graphics Driver

Command Buffer (Triangle Queue)

And, another program runs on each of these pixels (to determine, among others, what its color should be)

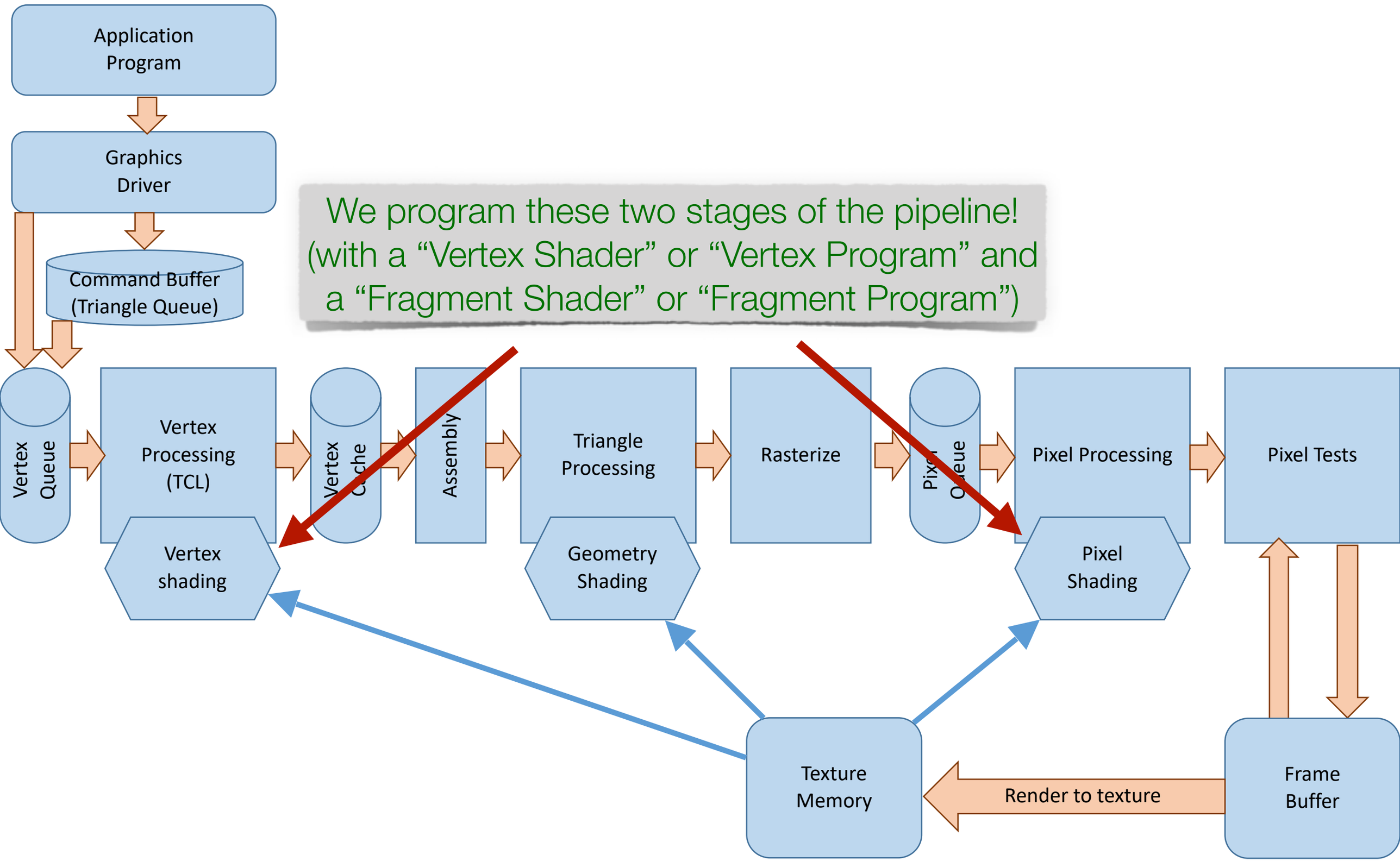


The (GPU) graphics pipeline



Some pixels are rejected (hidden, outside of view volume), the remaining ones written to the screen (really, the “framebuffer”)

The (GPU) graphics pipeline



The (GPU) graphics pipeline

Application Program

Graphics Driver

Command Buffer (Triangle Queue)

This program gets executed on
every vertex, of every triangle of our mesh

Vertex Queue

Vertex Processing (TCL)

Vertex shading

Vertex Cache

Assembly

Triangle Processing

Geometry Shading

Rasterize

Pixel Queue

Pixel Processing

Pixel Shading

Pixel Tests

Texture Memory

Render to texture

Frame Buffer

The (GPU) graphics pipeline