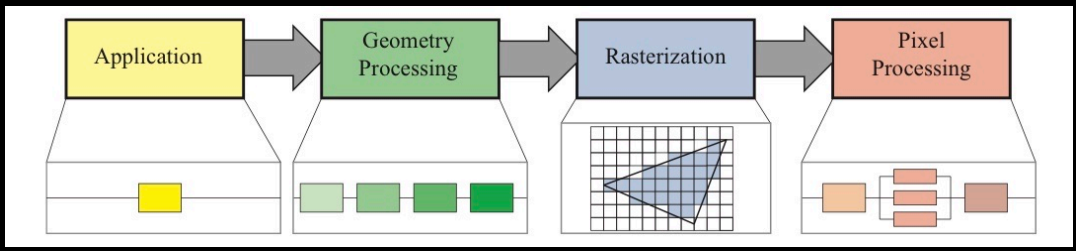
General Pipeline Architecture



Application Stage

usually execute on **CPU**

Developer has full control over this stage → could help later stages < less triangles >

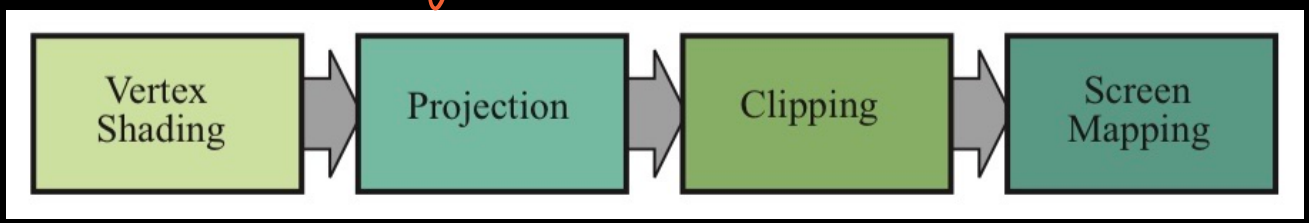
cannot divide into sub-stages, but tasks can be assigned to multi-cores.

render primitives

Collision Detection

input handling

Geometry Processing GPU



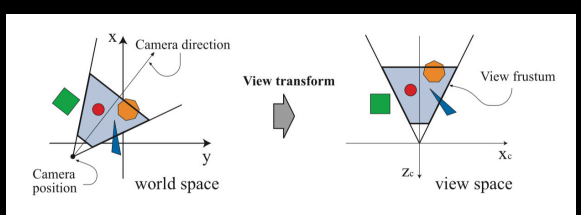
vertex shading

① compute the vertex position to world view space

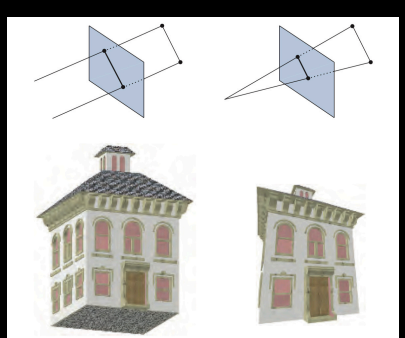
② operate on vertex data < normals, tex coord ... >

numerical data will be interpolated during rasterization

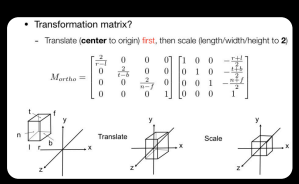
③ possible animation < vertex blending - deformation >



projection



Orthographic Perspective



ortho matrix = translate + scale

projection matrix 推算过来

先 $M_{persp \to ortho}$ 再 M_{ortho}

$M_{persp \to ortho}$:

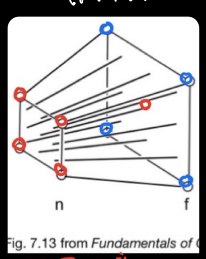


Fig. 7.13 from Fundamentals of

在near不变

z值不变

Premise ① NEAR 不变

Therefore $(0\ 0\ A\ B)\begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = An + B = n^2$

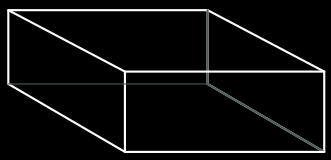
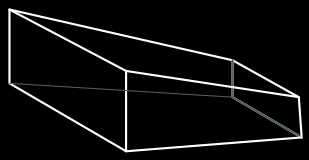
Premise ② CENTER OF FAR $\begin{pmatrix} 0 \\ 0 \\ f \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \\ 0 \\ f^2 \\ 1 \end{pmatrix}$ 不变

Therefore $(0\ 0\ A\ B)\begin{pmatrix} x \\ y \\ f \\ 1 \end{pmatrix} = Af + B = f^2$

$\begin{cases} An + B = n^2 \\ Af + B = f^2 \end{cases} \Rightarrow \begin{cases} A = n + f \\ B = -nf \end{cases}$

$$M_{persp \to ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Perspective Division divide by w 齐次除法 → Normalized Device Coordinate

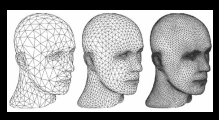

坐标系服务于

canonical view volume

models are now in clip coords

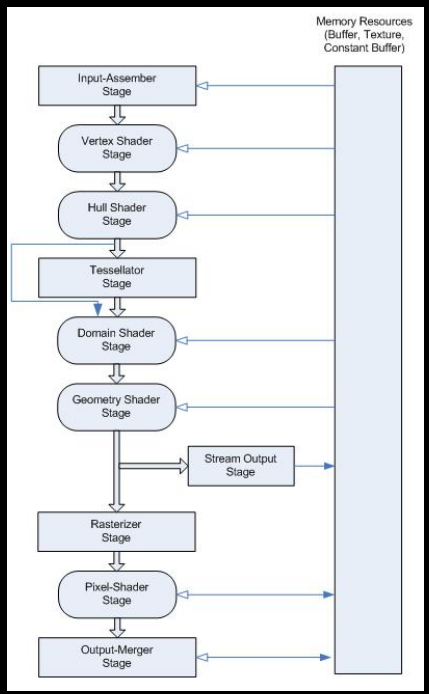
△ After this stage, z values are stored in a z-buffer

Optional Vertex Processing



Tessellation generates primitives on GPU (subdivision) after vertex shader

- hull shader: sets the type & number of subdivisions LOD
- tesselator: actually do the subdivision
- domain shader: called for each newly generated vertex
  - sets the type of primitive to generate/ways to space vertecies
  - can also do per-vertex calculation as a VS.



Geometry Shading also generates primitives, but limited in scopes and types. Commonly used in Particle Generation.

· →  provide convincing primitives for future shading.

∀ facing view point

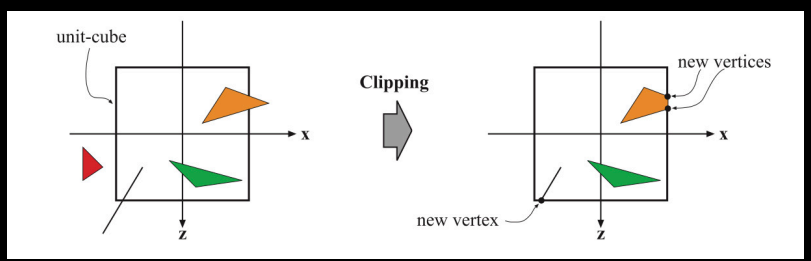
Stream Output allowing output processed vertices to CPU or GPU <later passes> as data array. Typically used for Particle Simulation.

Clipping

sending primitives inside view volume to rastrization stage

clip the primitives partially in the volume.

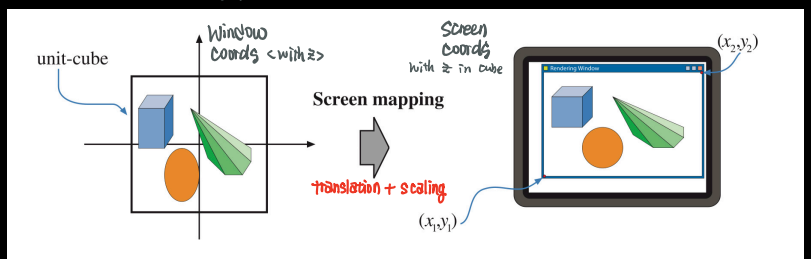
coordinates are still 3-dimensional before entering



Screen Mapping

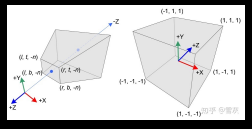
window-to-viewport transformation

attached on 1 camera, 1 cam can bind many viewports

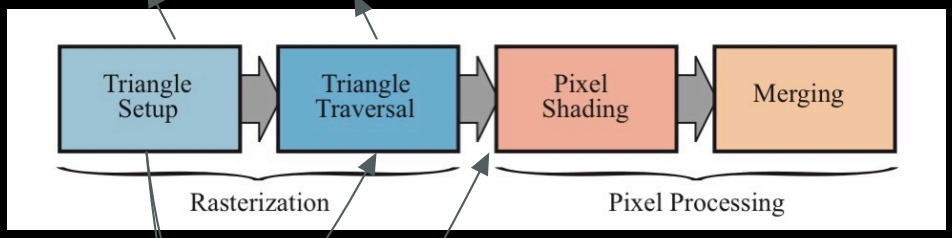


how to define texture data with int & float

$$d = \texttt{floor}(c),$$
$$c = d + 0.5,$$



Rasterization



differentials
edge equations
...
useful for future stages <interpolation>

performed by dedicated, hardwired silicons

perspective correction

screen space 插值不准确 → pespective divide 非线性

需要 correction

fragments with interpolated data

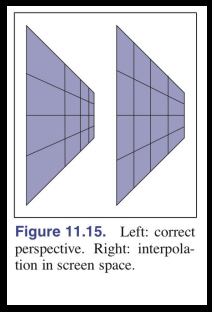


Figure 11.15. Left: correct perspective. Right: interpolation in screen space.

Pixel Shading

performed by programmable GPU cores

Implement techniques including texturing

color data

Merging

combine the color result from pixel shader with the color data already in the color buffer

Raster Operations <blend operations>

△ NOT typically programmable, but configurable, enabling various effects.

Resolving Visibility done with depth buffer O(n) comparison

△ transparent objects cannot be rendered with depth buff
they must be rendered after all opaque objects are rendered and done with back-to-front order.

alpha test was used to make sure transparent pixels don't affect z-buffer.
now alpha channel is controllable in pixel shader

stencil buffer special effects mask outline