Graphics Pipeline

Ahmed H. Mahmoud

17 January 2017

The graphics processing unit (GPU) is the hardware in every computer that is responsible of generating and displaying images on the computer screen. It is different from the CPU or Central Processing Unit which has different tasks like converting the keystrokes on the keyboard into commands and then executing these commands accordingly. The recent demand on stunning visuals like in video-games or animated movies has driven research community to optimize and improve the performance of the GPU. In this paper, we will describe the major steps that the GPU takes in order to create a scene on the screen. These steps are called Graphics Pipeline. We shall describe this process by making an analogy to how we, humans, display a new piece of cloth on a mannequin in a shop-window.

The story starts with an application running on the computer and asks to update the screen with new scene. Such an application runs by the CPU which commands the GPU and sends all necessary resources to the GPU to update or generate a new scene [2]. For example, if the application is a soccer video-game, the CPU will calculate the new position of the player and the ball. Then, it will send this information to the GPU and order it to update the scene. In our shop-window analogy, CPU is like the designer who has a new dress and wants to put it in display. In addition to sending the dress, the designer might also ask for a mannequin of certain size, information how it should be displayed, the lighting, etc. Looking closely to the mannequin, we find out that it composed of different pieces that are glued together e.g., the head is made of a sphere-like shape while hands and legs has a cylinder-ish shape. This is similar to the information CPU sends to the GPU to create the scene. It sends the scene in some simple primitives that can be put together to create some complex shapes. The primitives the computer uses are *vertices* (points). These vertices can be connected to together to

create triangles. Computer uses triangles because it is easy to break down any complex shape into triangles. Also, triangles has some nice mathematical properties that make the rest of the calculations simpler.

First stage in the pipeline is called *vertex processing* [1]. When the CPU send different objects (player, soccer ball, etc), each of the these objections has no information about where the other objects are. Thus, each object lives in it own world; known as *object* space. The vertex processing stage takes all these objects and places them in different space; *world* space. This may also include scaling or rotating of the objects. Back to shop-window, this is similar to taking our mannequin and placing it in the center of the window as commanded by the designer and placing different accessories around it. This may include also tilting the mannequin head to look cooler.

Next, we need to setup the lighting to lit up the vertices. For the shop-window, this is trivial. We will just put few spotlights with different colors and nature will work out its way to produce the lighting and the mannequin's shadow which may dim a piece of accessory a little bit. On computer, the GPU simulates what nature has done. So, the CPU sends the information about the lighting sources; its intensity, colors, positions, different surfaces material and how it correspond to light. GPU, using mathematical equations, figures out the lighting and shading per vertex. GPU has a wide range of flexibility from producing such photo-realistic scenes to assign a single color per vertex. The trade-off is between high-quality photo-realistic scenes and time consumption. This flexibility is termed differently in computer graphics; *programmable* pipeline stage.

Since the designer does not know the size of the shop-window, she may ask to place accessories that will not be visible from outside. The solution is to remove any accessories outside the visible scope. Additionally, any objects (e.g., part of mannequin body or accessories) that lie on the boundaries should be clipped out. What we end up with is the objects in *camera* space, since the eye looking from the outside is analogous to a camera. The GPU does the same operation; chopping out whatever will not be seen from the camera point of view. The result is the set of triangles that will eventually make it to the screen. This is called *clipping and primitive assembly*; since clipping a triangle may result into non-triangular shapes, so we need to re-triangulate such shapes.

So far we have been dealing with a 3D objects made of simplified primitives. But the computer monitor can only view 2D objects. Moreover, the screen display objects as a set of very small grid of dots called *pixels*. To

do this conversion, we overlay the 3D objects with a graph paper or squared wire mesh. The small square in the graph paper or wire mesh are called *fragments*. Each fragment will inherent different attributes of the fraction of the object it is overlaid on. A fragment that will make it to the end of the pipeline will be used to update a pixel on the screen. The stage of converting the triangles into fragment is called *rasterization*. Note that we overlay this wire mesh over all objects we end up with after clipping. Thus, a fragment will carry different information about the color, position and depth. The depth is used later to filter out some fragments that are far behind and thus covered by other fragments.

So far we have been processing the mannequin and everything around it, but we have not dressed it up yet. Unlike what we do in real life and since we deal with pixels now rather than 3D objects, we dress the fragment by cutting out the dress into pieces and stitching it to the right fragment. This is called *texture mapping*, where we map a texture to fragments. This is done in the final stage; *fragment processing*. Fragment processing also figures out which fragment will be visible and which will be hidden. Additionally, if a fragment is associated with translucent effects, blending effects will be created with the fragments behind. We can also add some fancy effects (e.g., smoke, fog) to the fragments which make computer graphics a very powerful tool. Similar to vertex processing stage, fragment processing is also programmable. The final set of fragments is sent to the screen pixels for display.

References

[1] E Angle and D Shreiner. Interactive computer graphics: A top-down approach with shader-based OpenGL, 2011.

[2] Li-Yi Wei. A crash course on programmable graphics hardware. *Microsoft Research Asia, Tsinghua University, Beijing*, 2005.