A standard rendering pipeline for polygon based object using Phong or Gouraud shading techniques have eight stages [1] First, all the potentially visible objects in the database (the virtual world) are selected. This enable the virtual world to have a time component, since objects can be added, deleted or relocated before the image is generated. The selection process also filter out the some of the non-visible objects, so the total amount rendering effort is reduced. A scene can be organized by a tree structure with the computer graphics object as the leaf nodes. The inner nodes represents a complex object by combining its children. Hence, the visibility of a object is based on the visibility of its parent. Since this tree structure is static for a particular image, the traversal for each leaf object can be doen in parallel.

Second step is to transform all the remaining object to their proper orientation in the virtual world. The transformation process uses a collection of 4×4 matrices for translation,

$$T(x,y,z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{0.1}$$

rotation on X, Y and Z axis,

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{0.2}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{0.3}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{0.4}$$

scaling

$$S(s_x,s_y,s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{0.5}$$

. Although rotation and scaling of a 3D object can be done by using 3×3 matrices, only 4×4 matrices can incorporate the translation operation. The coordinates, (x, y, z), of a

3D object is mapped into a 3D subspace, which has 4D coordinates, $(x, y, z, 1)$, of a 4D universe. The rotation and scaling is still operated within the 3D subspace, so an identity matrix is used for the 4th dimension.

These matrices can be multiplied together to form one single 4×4 matrix

$$U = T(x, y, z)R_x(\theta)R_y(\theta)R_z(\theta)S(s_x, s_y, s_z) \quad (0.6)$$

The combined matrix, U, is applied to each vertices of all the objects. The transformation of one vertex requires 16 floating-point multiplication and 12 floating-point addition. Since the transformation calculation only depends on the transformation matrix and the value of the vertex, all the vertices can be transformed in parallel.

Third, the objects are checked to see if they will appear in the image. The image is the only window for the viewer to peek into the virtual world. Similar to a real window the image can only show part of the virtual world. Therefore, some of the objects in the virtual world may not appear in the image. Eliminating these unseen objects reduce the total amount of effort needed to generate the image. A object is visible, if its volume intersect or contain within the visible zone of the virtual world. The intersection calculations can be CPU intensive, if the object have a complex shape. A transparent, simple bounding volume, such as a box or a sphere, which surrounds the object, can be use to simplify the calculation. However, the simple bounding volume is larger than the object, so some of the unseen objects will be marked as. Tighter bounding volumes imply a more accurate visible objects list. A trade off is made between the amount of work on intersection calculation and the amount of rendering effort saved, and the goal is to reduce the overall work to render the image. The visible calcualtion involves only the boundary of the objects and view volume. The visible calculation of each objects can be performed in parallel.

Forth, the color of the objects are computed by using the properties of the light(s) and the surface properties of the object. The properties of light consists of its position and color. The relative position of the light and the object determines the color intensity of the pixel. If the surface is facing away the light, then its color will be darker (less intense). The facing direction of the surface, where the pixel is represented, is encoded by the surface normal. A surface normal pointing toward the light indicates that the surface is illuminated. However, this does not take into the account of shadows casted by the near by objects. Omitting shadow is one of the many approximations to minimize rendering resources and time. If the surface is illuminated, then the color of the surface is a combination of the object's color and the color of the light, for example a red object under a blue light appears black. Since the color of a object only depends on the illumination and its surface properties, the color of the objects can be done concurrently.

Fifth, the objects are transformed according to the viewing specification. For terrain, a perspective view is typical. The perspective transformation,

$$M(d) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix}, \quad (0.7)$$

scales the objects inversely proportional to the distance, d, between the viewpoint and the objects. The visual effect is that all vertical parallel lines converge at a distance point and horizontal parallel lines remain parallel. Hence, the rendered image conformed to our perception in the real world, where the distance objects appear smaller but not deformed. The numerical operation of this step is the same as the transformation step, so all view transformation can be executed in parallel.

Sixth, the polygon based objects are clipped according to the view volume. The invisible objects, which escaped from step three, will be eliminated entirely. Objects that are behind the viewer, too close or too far are eliminated. If a object behind the viewer is rendered, then it will have a mirror reflected image because d is negative and the negative sign will propagate to its coordinate. Removing the objects that are too close will avoid numerical problem in the next stage of the pipeline. Objects do not contribute to the image, so removing them will reduce the amount of rendering effort. In addition, partially visible objects will have new borders that flush to the side of the view volume. New polygons are formed by removing the invisible portion of original polygon. The clipping computation is done by using line and plane intersection calculation. Three type of parallelism, polygon, edge, and vertices, can be exploit here. Each of these data elements is a self-contains unit and there is no other influence that need to take into account, so the work associate with each type of data elements can be done in parallel.

Seventh, the objects are mapped back to the initial 3D space. The operating principal for the view transformation is to identifies the corresponding points in the other 3D subspaces for all points in initial 3D subspace. After the view transformation, the coordinates of the objects have the form of (x, y, Z, w). The object is mapped back to the 3D world by dividing z, y and z with w. The process of projecting the 3D objects onto a 2D plane is similar to the process of taking a photograph of the objects. For a polygon based object, the projecting process consists of ignoring the z coordinate of all vertices and scaling the 2D object to the size specified by the image size. The mapping of each vertices can be done in parallel because of the independency of the vertices.

Finally, the 2D objects are drawn onto the image buffer pixel by pixel, which is called rasterization. The rasterization of the 3D objects is the same as taking a photograph of the objects by ignoring the z coordinate of all the vertices. However, the 2D objects may overlap each other on the 2D plane because the distance 3D objects may be located behind the closer 3D objects. An arbitrary order of rasterizing the 2D objects may put the distance objects in front of a closer object. The correct order of rasterization can be retrieved by sorting the list of visible objects based on the z coordinate. Another popular method, which is called Z-buffer method, is to record the depth information along with the color value for each pixel. Only the pixel closest to the projection plane is stored, and the pixel contributed by objects further away will be ignored. As a result, a correct image can be generated even with an arbitrary rasterization order. For the depth sorting rendering, the polygon parallelism can be exploited for the polygons at the same depth level. For Z-buffer rendering, all polygon can be rasterized in parallel because the depth value is computed for each pixel. However, for both rasterization methods, pixel parallelism can be exploited beacuse the calculation for color of each pixel only depends on the color of the conbrbuting polygon.

Assuming rasterization are done with Z-buffer and polygon vertices are in clockwise order in the polygon's vertices list,

forall visible polygon do

```
    find the starting scanline
    locate the left and right edge
    while the left edge do not past or coinside the right edge
        forall pixel between the left and right edge
            compute their current depth
            where (current depth < recorded depth)
                computer their colors
        advance to next scanline
        where next scanline landed on or just pasted a vertex
            another vertex on the same side is found
MPI_reduce(MPI_COMM_WORLD, image_buffer, MPI_ZBUFFER)
```

There two potential bottlenecks, memory bus bandwidth, communication bandwidth and contension, in implementing a parallel rendering algorithm . The memory bus banwidth bottlenecks comes from moving pixel information to and from communication buffers during the construction of the final image. This is demonstrated by all special graphics rendering hardwares have decated hardware to perform memory to memory copy. After the pixel is moved to the communication buffer, the information is sent through communication channels that connects the computing nodes. Depending on the connection topology and the communication pattern, it is possible to have contension for the channels. Furthermore, the channels has a limited throughput, so the arrival of the data is delayed.

Bibliography

[1] James D. Foley, Andries van Dam, Stenve K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 2nd edition, 1992.