Chapter 8

z-BUFFER, GOURAUD-SHADING WORKSTATIONS

As different shading methods and visibility calculations have diversified the image generation, many different alternatives have come into existence for their implementation. This chapter will focus on a very popular solution using the z-buffer technique for hidden surface removal, and Gouraud shading for color computation.

The main requirements of an advanced workstation of this category are:

- The workstation has to generate both 2D and 3D graphics at the speed required for interactive manipulation and real-time animation.
- At least wire-frame, hidden-line and solid — Gouraud and constant shaded — display of 3D objects broken down into polygon lists must be supported. Some technique has to be applied to ease interpretation of wire frame images.
- Both parallel and perspective projections are to be supported.
- Methods reducing the artifacts of sampling and quantization are needed.
- The required resolution is over 1000×1000 pixels, the frame buffer must have at least 12, but preferably 24 bits/pixel to allow for true

color mode and double buffering for animation. The z-buffer must have at least 16 bits/pixel.

8.1 Survey of wire frame image generation

The dataflow model of the wire frame image generation in a system applying z-buffer and Gouraud shading is described in figure 8.1. The **decomposition** reads the internal model and converts it to a wire-frame representation providing a list of edges defined by the two endpoints in the local modeling coordinate system for each object. The points are transformed first by the **modeling transformation** $\mathbf{T_M}$ to generate the points in the common world coordinate system. The modeling transformation is set before processing each object. From the world coordinate system the points are transformed again to the screen coordinate system for parallel projection and to the 4D homogeneous coordinate system for perspective projection by a **viewing transformation** $\mathbf{T_V}$. Since the matrix multiplications needed by the modeling and viewing transformations can be concatenated, the transformation from the local modeling coordinates to the screen or to the 4D homogeneous coordinate system can be realized by a single matrix multiplication by a composite transformation matrix $\mathbf{T_C} = \mathbf{T_M} \cdot \mathbf{T_V}$.

For parallel projection, the complete clipping is to be done in the screen coordinate system by, for example, the 3D version of the Cohen–Sutherland clipping algorithm. For perspective projection, however, at least the **depth clipping** phase must be carried out before the homogeneous division, that is in the 4D homogeneous coordinate system, then the real 3D coordinates have to be generated by the homogeneous division, and **clipping against the side faces** should be accomplished if this was not done in the 4D homogeneous space.

The structure of the screen coordinate system is independent of the type of projection, the X, Y coordinates of a point refer to the projected coordinates in pixel space, and Z is a monotonously increasing function of the distance from the camera. Thus the **projection** is trivial, only the X, Y coordinates have to be extracted.

The next phase of the image generation is **scan conversion**, meaning the selection of those pixels which approximate the given line segment and also the color calculation of those pixels. Since pixels correspond to the integer

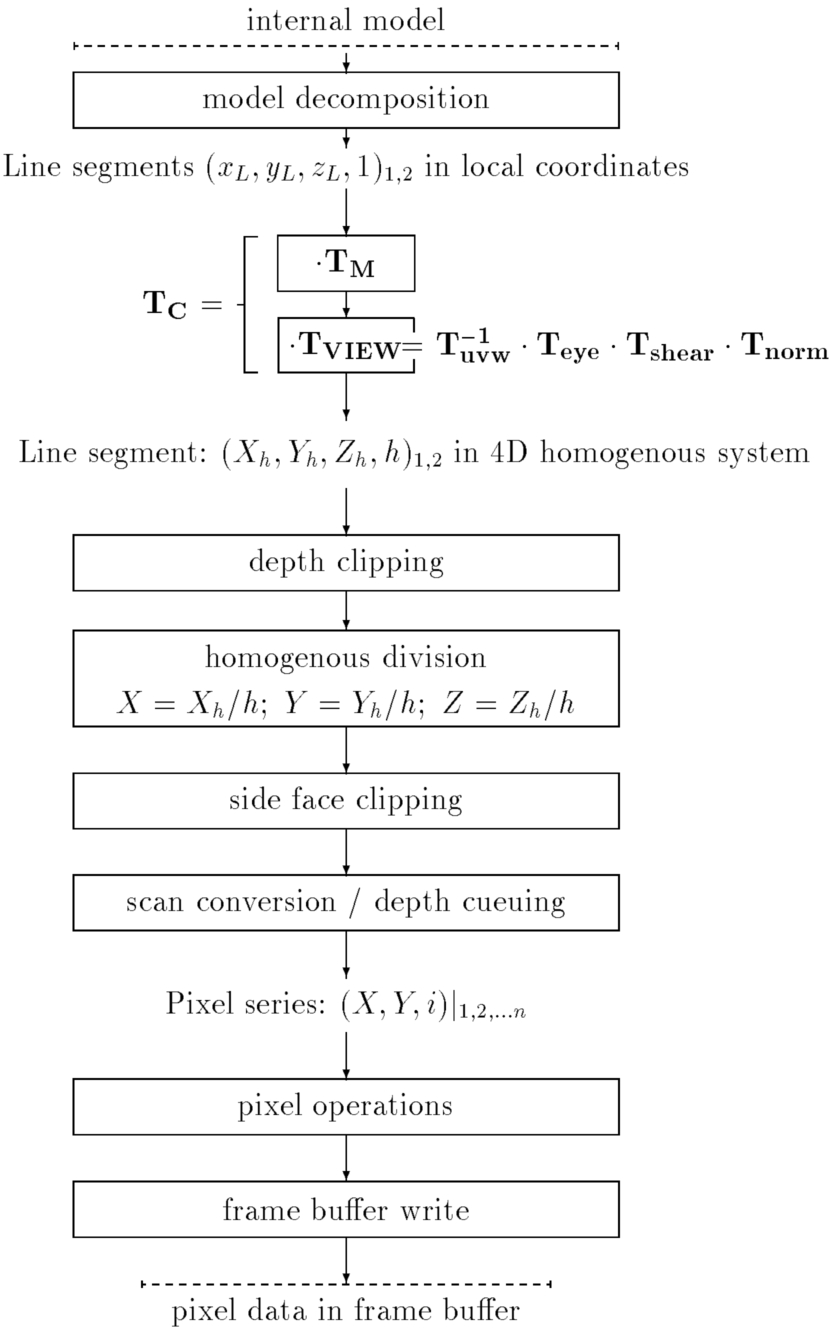

Figure 8.1: Data flow of wire frame image synthesis (perspective projection)

grid of pixel space, and scan conversion algorithms usually rely on the integer representation of endpoint coordinates, the coordinates are truncated or rounded to integers.

Concerning color calculation, or **shading**, it is not worth working with sophisticated shading and illumination models when the final image is wire-frame. The simple assumption that all pixels of the vectors have the same color, however, is often not satisfactory, because many lines crossing each other may confuse the observer, inhibiting reconstruction of the 3D shape in his mind. The understandability of wire-frame images, however, can be improved by a useful trick, called **depth cueing**, which uses more intense colors for points closer to the camera, while the color decays into the background as the distance of the line segments increases, corresponding to a simplified shading model defining a single lightsource in the camera position.

The outcome of scan-conversion is a series of pixels defined by the integer coordinates X_p, Y_p and the pixel color i. Before writing the color information of the addressed pixel into the raster memory various operations can be applied to the individual pixels. These **pixel level operations** may include the reduction of the quantization effects by the means of **dithering**, or arithmetic and logic operations with the pixel data already stored at the X_p, Y_p location. This latter procedure is called the **raster operation**.

Anti-aliasing techniques, for example, require the weighted addition of the new and the already stored colors. A simple exclusive OR (XOR) operation, on the other hand, allows the later erasure of a part of the wire-frame image without affecting the other part, based on the identity $(A \oplus B) \oplus B = A$. Raster operations need not only the generated color information, but also the color stored in the frame buffer at the given pixel location, thus an extra frame buffer read cycle is required by them.

The result of pixel level operations is finally written into the **frame buffer memory** which is periodically scanned by the video display circuits which generate the color distribution of the display according to the stored frame buffer data.

8.2 Survey of shaded image generation

The dataflow model of the shaded image generation in a **z-buffer**, **Gouraud shading** system is described in figure 8.2.

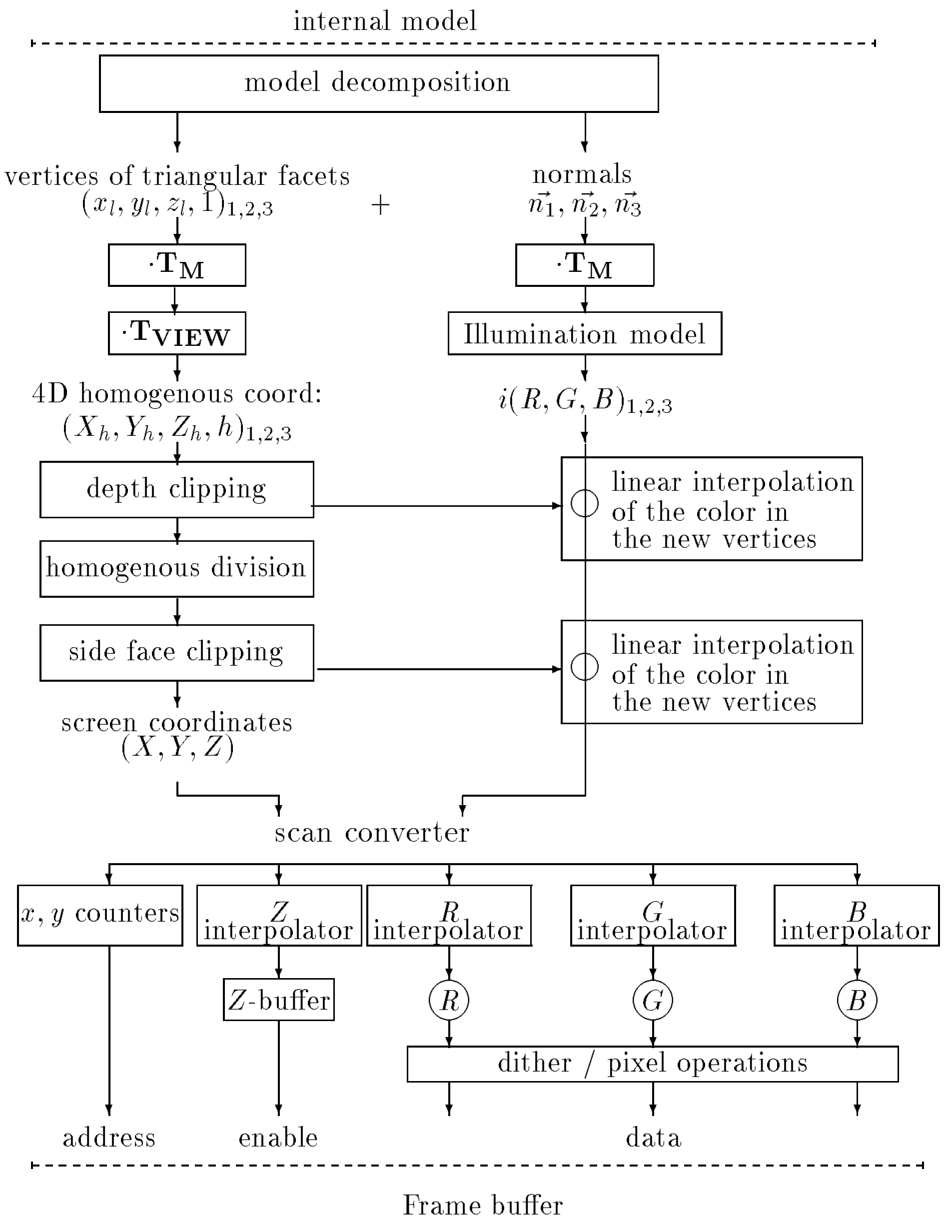

Figure 8.2: Data flow of shaded image synthesis

Now the **decomposition** reads the internal model and converts it to a polygon list representation defining each polygon by its vertices in the local modeling coordinate system for each object. To provide the necessary information for shading, the real normals of the surfaces approximated by polygon meshes are also computed at polygon vertices. The vertices are transformed first by the modeling transformation then by the viewing transformation by a single matrix multiplication with the composite transformation matrix. Normal vectors, however, are transformed to the world coordinate system, because that is a proper place for illumination calculation. Coordinate systems after shearing and perspective transformation are not suitable, since they do not preserve angles, causing incorrect calculation of dot products. According to the concept of Gouraud shading, the **illumination equation** is evaluated for each vertex of the polygon mesh approximating the real surfaces, using the real surface normals at these points. **Depth cueing** can also be applied to shaded image generation if the illumination equation is modified to attenuate the intensity proportionally to the distance from the camera. The linear decay of the color at the internal pixels will be guaranteed by linear interpolation of the Gouraud shading.

Similarly to wire frame image generation, the complete clipping is to be done in the screen coordinate system for parallel projection. An applicable clipping algorithm is the 3D version of the Sutherland-Hodgman polygon clipping algorithm. For perspective projection, however, at least the **depth clipping** phase must be done before homogeneous division, that is in the 4D homogeneous coordinate system, then the real 3D coordinates have to be generated by homogeneous division, and **clipping against the side faces** should be accomplished if this was not done in 4D homogeneous space.

After the trivial **projection** in the screen coordinate system, the next phase of image generation is **scan conversion** meaning the selection of those pixels which approximate the given polygon and also the interpolation of pixel colors from the vertex colors coming from the illumination formulae evaluated in the world coordinate system. Since pixels correspond to the integer grid of the pixel space, and scan conversion algorithms usually rely on the integer representation of endpoint coordinates, the coordinates are truncated or rounded to integers. The z-buffer visibility calculation method resolves the hidden surface problem during the scan conversion comparing the Z-coordinate of each pixel and the value already stored in the z-buffer memory. Since the transformation to the screen coordinate system has been carefully selected to preserve planes, the Z-coordinate of an inner point can be determined by linear interpolation of the Z-coordinates of the vertices. This Z-interpolation and the color interpolation for the R, G and B components are usually executed by a digital network. Since in hardware implementations the number of variables is not flexible, polygons must be decomposed into triangles defined by three vertices before the interpolation.

The pixel series resulting from the polygon or **facet** scan conversion can also go through pixel level operations before being written into the frame buffer. In addition to dithering and arithmetic and logic raster operations, the illusion of transparency can also be generated by an appropriate pixel level method which is regarded as the application of **translucency patterns**. The final colors are eventually written into the frame buffer memory.

8.3 General system architecture

Examining the tasks to be executed during image generation from the point of view of data types, operations, speed requirements and the allocated hardware resources, the complete pipeline can be broken down into the following main stages:

1. *Internal model access and primitive decomposition.* This stage should be as flexible as possible to incorporate a wide range of models. The algorithms are also general, thus some general purpose processor must be used to run the executing programs. This processor will be called the **model access processor** which is a sort of interface between the graphics subsystem and the rest of the system. The model access and primitive decomposition step needs to be executed once for an interactive manipulation sequence and for animation which are the most time critical applications. Thus, if there is a temporary memory to store the primitives generated from the internal model, then the speed requirement of this stage is relatively modest. This buffer memory storing graphics primitives is usually called the **display list memory**. The display list is the low level representation of the model to be rendered on the computer screen in conjunction with the camera and display parameters. Display lists are interpreted and processed by a so-called **display list processor** which controls the functional

elements taking part in the image synthesis. Thus, the records of display lists can often be regarded as operation codes or instructions to a special purpose processor, and the content of the display list memory as an executable program which generates the desired image.

2. *Geometric manipulations* including transformation, clipping, projection and illumination calculation. This stage deals with geometric primitives defined by points represented by coordinate triples. The coordinates are usually floating point numbers to allow flexibility and to avoid rounding errors. At this stage fast, but simple floating point arithmetic is needed, including addition, multiplication, division and also square roots for shading calculations, but the control flow is very simple and there is no need for accessing large data structures. A cost effective realization of this stage may contain floating point signal processors, bit-slice ALUs or floating point co-processors. The hardware unit responsible for these tasks is usually called the **geometry engine**, although one of its tasks, the illumination calculation, is not a geometric problem. The geometry engines of advanced workstations can process about 1 million points per second.

3. *Scan-conversion, z-buffering and pixel level operations.* These tasks process individual pixels whose number can exceed 1 million for a single image. This means that the time available for a single pixel is very small, usually several tens of nanoseconds. Up to now commercial programmable devices have not been capable of coping with such a speed, thus the only alternatives were special purpose digital networks, or high degree parallelization. However, recently very fast RISC processors optimized for graphics have appeared, implementing internal parallelization and using large cache memories to decrease significantly the number of memory cycles to fetch instructions. A successful representative of this class of processors is the *intel 860* microprocessor [Int89] [DRSK92] which can be used not only for scan conversion, but also as a geometry engine because of its appealing floating point performance. At the level of scan-conversion, z-buffering and pixel operations, four sub-stages can be identified. **Scan conversion** is responsible for the change of the representation from geometric to pixel. The hardware unit executing this task is called the **scan converter**.

The **z-buffering hardware** includes both the comparator logic and the **z-buffer** memory, and generates an enabling signal to overwrite the color stored in the frame buffer while it is updating the z-value for the actual pixel. Thus, to process a single pixel, the z-buffer memory needs to be accessed for a read and an optional write cycle. Comparing the speed requirements — several tens of nanosecond for a single pixel —, and the cycle time of the memories which are suitable to realize several megabytes of storage — about a hundred nanoseconds —, it becomes obvious that some special architecture is needed to allow the read and write cycles to be accomplished in time. The solutions applicable are similar to those used for frame buffer memory design. Pixel level operations can be classified according to their need of color information already stored in the frame buffer. Units carrying out **dithering** and generating translucency patterns do not use the colors already stored at all. **Raster operations**, on the other hand, produce a new color value as a result of an operation on the calculated and the already stored colors, thus they need to access the frame buffer.

4. *Frame buffer* storage. Writing the generated pixels into the **frame buffer memory** also poses difficult problems, since the cycle time of commercial memories are several times greater than the expected few tens of nanoseconds, but the size of the frame buffer — several megabytes — does not allow for the usage of very high speed memories. Fortunately, we can take advantage of the fact that pixels are generated in a coherent way by image synthesis algorithms; that is if a pixel is written into the memory the next one will probably be that one which is adjacent to it. The frame buffer memory must be separated into **channels**, allocating a separate bus for each of them in such a way that on a scan line adjacent pixels correspond to different channels. Since this organization allows for the parallel access of those pixels that correspond to different channels, this architecture approximately decreases the access time by a factor of the number of channels for coherent accesses.

5. *The display of the content of the frame buffer* needs **video display hardware** which scans the frame buffer 50, 60 or 70 times each second

and produces the analog R, G and B signals for the color monitor. Since the frame buffer contains about 10^6 number of pixels, the time available for a single pixel is about 10 nanoseconds. This speed requirement can only be met by special hardware solutions. A further problem arises from the fact that the frame buffer is a double access memory, since the image synthesis is continuously writing new values into it while the video hardware is reading it to send its content to the color monitor. Both directions have critical timing requirements — ten nanoseconds and several tens of nanoseconds — higher than would be provided by a conventional memory architecture. Fortunately, the display hardware needs the pixel data very coherently, that is, pixels are accessed one after the other from left to right, and from top to bottom. Using this property, the frame buffer row being displayed can be loaded into a shift register which in turn rolls out the pixels one-by-one at the required speed and without accessing the frame buffer until the end of the current row. The series of consecutive pixels may be regarded as addresses of a **color lookup table** to allow a last transformation before **digital-analog conversion**. For indexed color mode, this lookup table converts the color indices (also called **pseudo-colors**) into R, G, B values. For **true color mode**, on the other hand, the R, G, B values stored in the frame buffer are used as three separate addresses in three **lookup tables** which are responsible for **γ-correction**. The size of these lookup tables is usually modest — typically $3 \times 256 \times 8$ bits — thus very high speed memories having access times less than 10 nanoseconds can be used. The outputs of the lookup tables are converted to analog signals by three **digital-to-analog converters**.

Summarizing, the following hardware units can be identified in the graphics subsystem of an advanced workstation of the discussed category: model access processor, display list memory, display list processor, geometry engine, scan converter, z-buffer comparator and controller, z-buffer memory, dithering and translucency unit, **raster operation ALUs**, frame buffer memory, video display hardware, lookup tables, D/A converters. Since each of these units is responsible for a specific stage of the process of the image generation, they should form a pipe-line structure. Graphics subsystems generating the images are thus called as the **output or image generation**

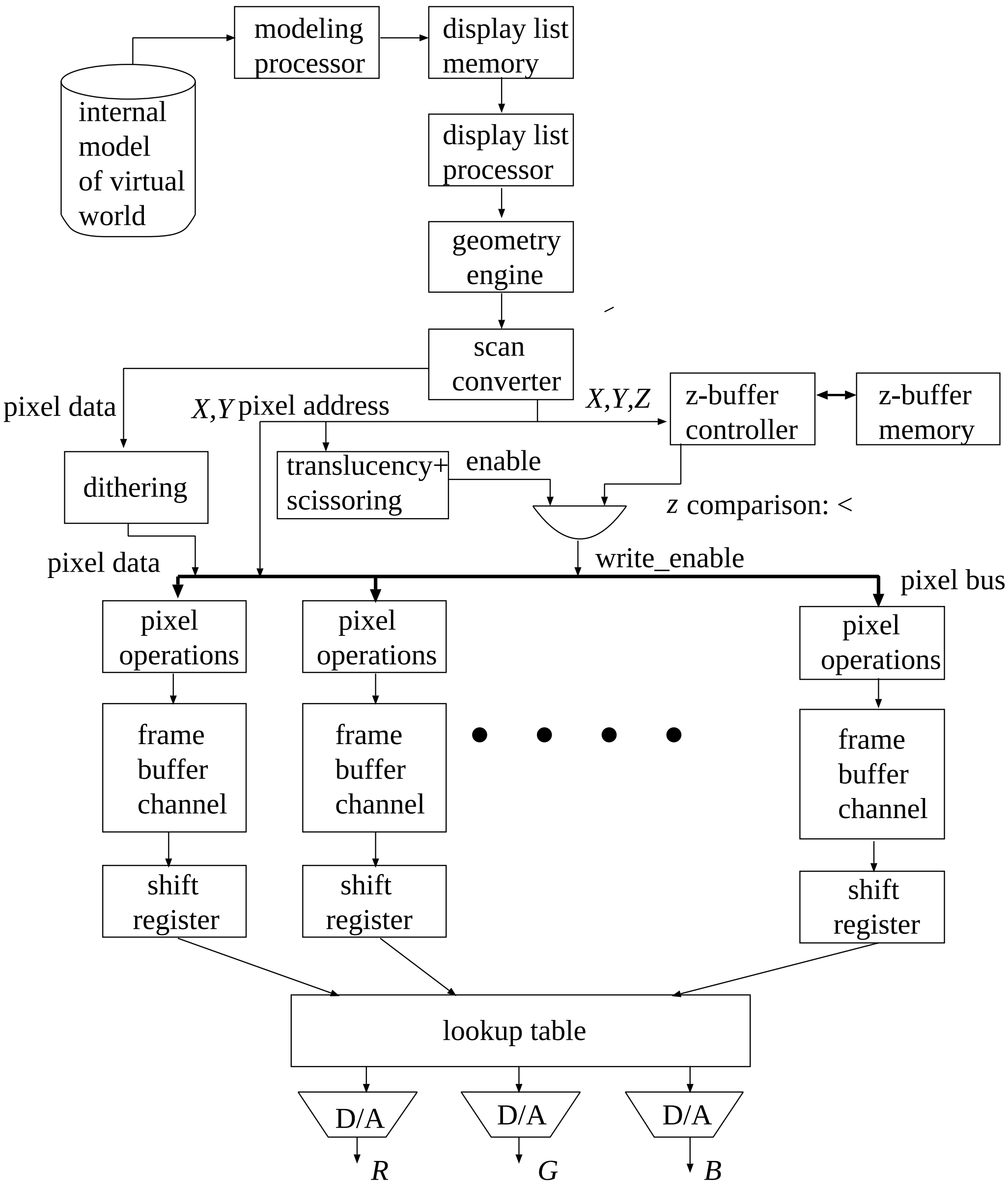

Figure 8.3: Architecture of z-buffer, Gouraud-shading graphics systems

pipelines. Interaction devices usually form a similar structure, which is called the **input pipeline**.

In the output pipeline the units can be grouped into two main subsystems: a high-level subsystem which works with geometric information and a low-level subsystem which deals with pixel information.

8.4 High-level subsystem

The high-level subsystem consists of the model access and display list processors, the display list memory and the geometry engine.

The model access processor is always, the display processor is often, a general purpose processor. The display list processor which is responsible for controlling the rest of the display pipeline can also be implemented as a special purpose processor executing the program of the display list. The display list memory is the interface between the model access processor and the display list processor, and thus it must have double access organization. The advantages of display list memories can be understood if the case of an animation sequence is considered. The geometric models of the objects need to be converted to display list records or instructions only once before the first image. The same data represented in an optimal way can be used again for each frame of the whole sequence, the model access processor just modifies the transformation matrices and viewing parameters before triggering the display list processor. Thus, both the computational burden of the model access processor and the communication between the model access and display list processors are modest, allowing the special purpose elements to utilize their maximum performance.

The display list processor interprets and executes the display lists by either realizing the necessary operations or by providing control to the other hardware units. A lookup table set instruction, for example, is executed by the display list processor. Encountering a *DRAWLINE* instruction, on the other hand, it gets the geometry engine to carry out the necessary transformation and clipping steps, and forces the scan converter to draw the screen space line at the points received from the geometry engine. Thus, the geometry engine can be regarded as the floating-point and special instruction set co-processor of the display list processor.

8.5 Low-level subsystem

8.5.1 Scan conversion hardware

Scan conversion of lines

The most often used line generators are the implementations of Bresenham's incremental algorithm that uses simple operations that can be directly implemented by combinational elements and does not need division and other complicated operations during initialization. The basic algorithm can generate the pixel addresses of a 2D digital line, therefore it must be extended to produce the Z coordinates of the internal pixels and also their color intensities if depth cueing is required. The Z coordinates and the pixel colors ought to be generated by an incremental algorithm to allow for easy hardware implementation. In order to derive such an incremental formula, the increment of the Z coordinate and the color is determined. Let the 3D screen space coordinates of the two end points of the line be $[X_1, Y_1, Z_1]$ and $[X_2, Y_2, Z_2]$, respectively and suppose that the z-buffer can hold values in the range $[0 \ldots Z_{\max}]$. Depth cueing requires the attenuation of the colors by a factor proportional to the distance from the camera, which is represented by the Z coordinate of the point. Assume that the intensity factor of depth cueing is $C_{\max}$ for $Z = 0$ and $C_{\min}$ for $Z_{\max}$. The number of pixels composing this digital line is:

$$L = \max\{|X_2 - X_1|, |Y_2 - Y_1|\}. \tag{8.1}$$

Since Z varies linearly along the line, the difference of the Z coordinates of two consecutive pixel centers is:

$$\Delta Z = \frac{Z_2 - Z_1}{L}. \tag{8.2}$$

Let I stand for any of the line's three color coordinates R, G, B. The perceived color, taking into account the effect of depth cueing, is:

$$I^*(Z) = I \cdot C(Z) = I \cdot (C_{\max} - \frac{C_{\max} - C_{\min}}{Z_{\max}} \cdot Z). \tag{8.3}$$

The difference in color of the two pixel centers is:

$$\Delta I = \frac{I^*(Z_2) - I^*(Z_1)}{L}. \tag{8.4}$$

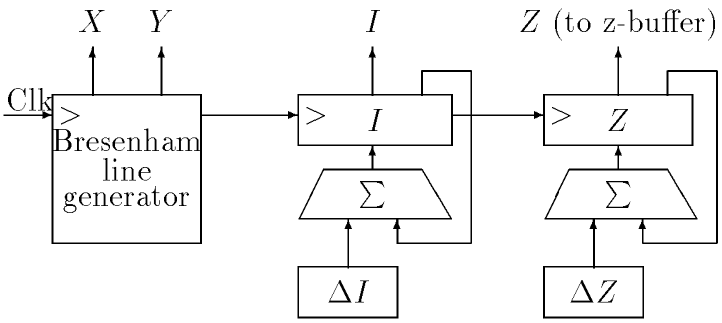

Figure 8.4: Hardware to draw depth cueing lines

For easy hardware realization, Z and I^* should be computed by integer additions. Examining the formulae for ΔZ and ΔI^*, we will see that they are non-integers and not necessarily positive. Thus, some signed fixed point representation must be selected for storing Z and I^*. The calculation of the Z coordinate and color I^* can thus be integrated into the internal loop of the Bresenham's algorithm:

3D_BresenhamLine $(X_1, Y_1, Z_1, X_2, Y_2, Z_2, I)$
    Initialize a 2D Bresenham's line generator(X_1, Y_1, X_2, Y_2);
    $L = \max\{|X_2 - X_1|, |Y_2 - Y_1|\}$;
    $\Delta Z = (Z_2 - Z_1)/L$;
    $\Delta I = I \cdot ((C_{\min} - C_{\max}) \cdot (Z_2 - Z_1))/(Z_{\max} \cdot L)$;
    $Z = Z_1 + 0.5$;
    $I^* = I \cdot (C_{\max} - (Z_1 \cdot (C_{\max} - C_{\min}))/Z_{\max}) + 0.5$;
    for $X = X_1$ **to** X_2 **do**
        Iterate Bresenham's algorithm(X, Y);
        I^* += ΔI; Z += ΔZ; $z = \mathrm{Trunc}(Z)$;
        if Zbuffer$[X, Y] > z$ **then**
            Write Zbuffer(X, Y, z);
            Write frame buffer$(X, Y, \mathrm{Trunc}(I^*))$;
        endif
    endfor

The z-buffer check is only necessary if the line drawing is mixed with shaded image generation, and it can be neglected when the complete image is wire frame.

Scan-conversion of triangles

For hidden surface elimination the z-buffer method can be used together with **Gouraud shading** if a shaded image is needed or with constant shading if a **hidden-line** picture is generated. The latter is based on the recognition that hidden lines can be eliminated by a special version of the z-buffer hidden surface algorithm which draws polygons generating their edges with the line color and filling their interior with the color of the background. In the final result the edges of the visible polygons will be seen, which are, in fact, the visible edges of the object. **Constant shading**, on the other hand, is a special version of the linear interpolation used in Gouraud shading with zero color increments. Thus the linear color interpolator can also be used for the generation of constant shaded and hidden-line images. The linear interpolation over a triangle is a two-dimensional interpolation over the pixel coordinates X and Y, which can be realized by a digital network as discussed in subsection 2.3.2 on hardware realization of multi-variate functions. Since a color value consists of three scalar components — the R, G and B coordinates — and the internal pixels' Z coordinates used for z-buffer checks are also produced by a linear interpolation, the interpolator must generate four two-variate functions. The applicable incremental algorithms have been discussed in section 6.3 (z-buffer method) and in section 7.5 (Gouraud shading). The complete hardware system is shown in figure 8.5.

8.5.2 z-buffer

The z-buffer consists of a Z-comparator logic and the memory subsystem. As has been mentioned, the memory must have a special organization to allow higher access speed than provided by individual memory chips when they are accessed coherently; that is in the order of subsequent pixels in a single pixel row. The same memory design problem arises in the context of the frame buffer, thus its solution will be discussed in the section of the frame buffer.

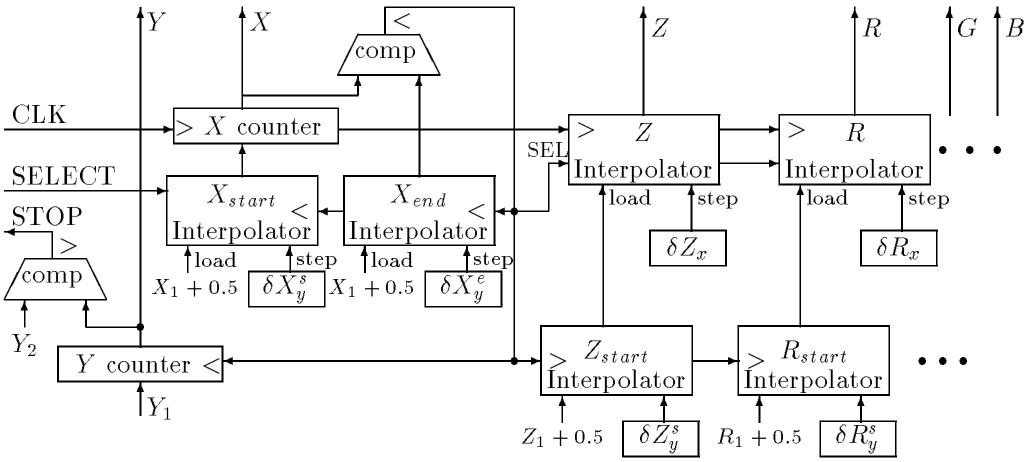

Figure 8.5: Scan converter for rendering triangles

The Z-comparator consists of a comparator element and a temporary register to hold the Z value already stored in the z-buffer. A comparison starts by loading the Z value stored at the X, Y location of the z-buffer into the temporary register. This is compared with the new Z value, resulting in an enabling signal that is true (enabled) if the new Z value is smaller than the one already stored. The Z-comparator then tries to write the new value into the z-buffer controlled by the enabling signal. If the enabling signal is true, then the write operation will succeed, otherwise the write operation will not alter the content of the z-buffer. The same enabling signal is used to enable or disable rewriting the content of the frame buffer to make the z-buffer algorithm complete.

8.5.3 Pixel-level operation

There are two categories of pixel-level operations: those which belong to the first category are based on only the new color values, and those which generate the final color from the color coming from the scan converter and the color stored in the frame buffer fall into the second category. The first category is a post-processing step of the scan conversion, while the second is a part of the frame buffer operation. Important examples of the post-processing class are the transparency support, called the **translucency generator**, the dithering hardware and the **overlay management**.

Support of translucency and dithering

As has been stated, transparency can be simulated if the surfaces are written into the frame buffer in order of decreasing distance from the camera and when a new pixel color is calculated, a weighted sum is computed from the new color and the color already stored in the frame buffer. The weight is defined by the transparency coefficient of the object. This is obviously a pixel operation. The dependence on the already stored color value, however, can be eliminated if the weighting summation is not restricted to a single pixel, and the low-pass filtering property of the human eye is also taken into consideration.

Suppose that when a new surface is rendered some of its spatially uniformly selected pixels are not written into the frame buffer memory. The image will contain pixel colors from the new surface and from the previously rendered surface — which is behind the last surface — that are mixed together. The human eye will filter this image and will produce the perception of some mixed color from the high frequency variations due to alternating the colors of several surfaces.

This is similar to looking through a fine net. Since in the holes of the net the world behind the net is visible, if the net is fine enough, the observer will have the feeling that he perceives the world through a transparent object whose color is determined by the color of the net, and whose transparency is given by the relative size of the holes in the net.

The implementation of this idea is straightforward. Masks, called **translucency patterns**, are defined to control the effective degree of transparency (the density of the net), and when a surface is written into the frame buffer, the X, Y coordinates of the actual pixel are checked whether or not they select a 0 (a hole) in the mask (net), and the frame buffer write operation is enabled or disabled according to the mask value.

This check is especially easy if the mask is defined as a 4×4 periodic pattern. Let us denote the low 2 bits of X and Y coordinates by $X|_2$ and $Y|_2$ respectively. If the 4×4 translucency pattern is $T[x, y]$, then the bit enabling the frame buffer write is $T[X|_2, Y|_2]$.

The hardware generating this can readily be combined with the dithering hardware discussed in subsection 11.5.2 (ordered dithers), as described in figure 8.6.

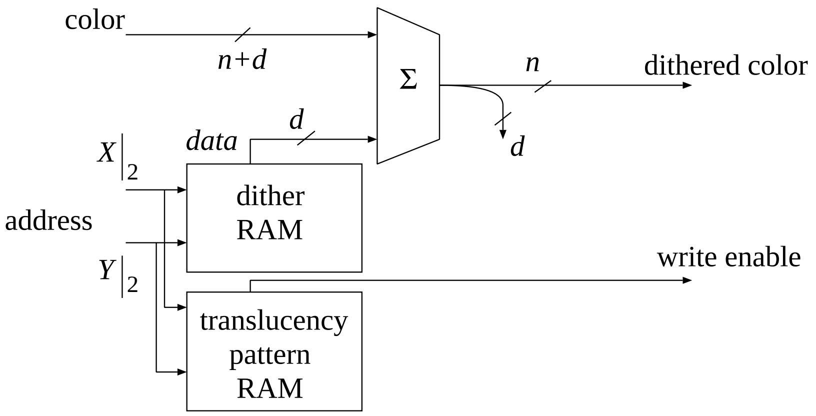

Figure 8.6: Dither and translucency pattern generator

8.5.4 Frame buffer

The frame buffer memory is responsible for storing the generated image in digital form and for allowing the video display hardware to scan it at the speed required for flicker-free display. As stated, the frame buffer is a double access memory, since it must be modified by the drawing engine on the one hand, while it is being scanned by the video display hardware on the other hand. Both access types have very critical speed requirements which exceed the speed of commercial memory chips, necessitating special architectural solutions. These solutions increase the effective access speed for "coherent" accesses, that is for those consecutive accesses which need data from different parts of the memory. The problem of the video refresh access is solved by the application of temporary shift registers which are loaded parallelly, and are usually capable of storing a single row of the image. These shift registers can then be used to produce the pixels at the speed of the display scan (approx. 10 nsec/pixel) without blocking memory access from the drawing engine.

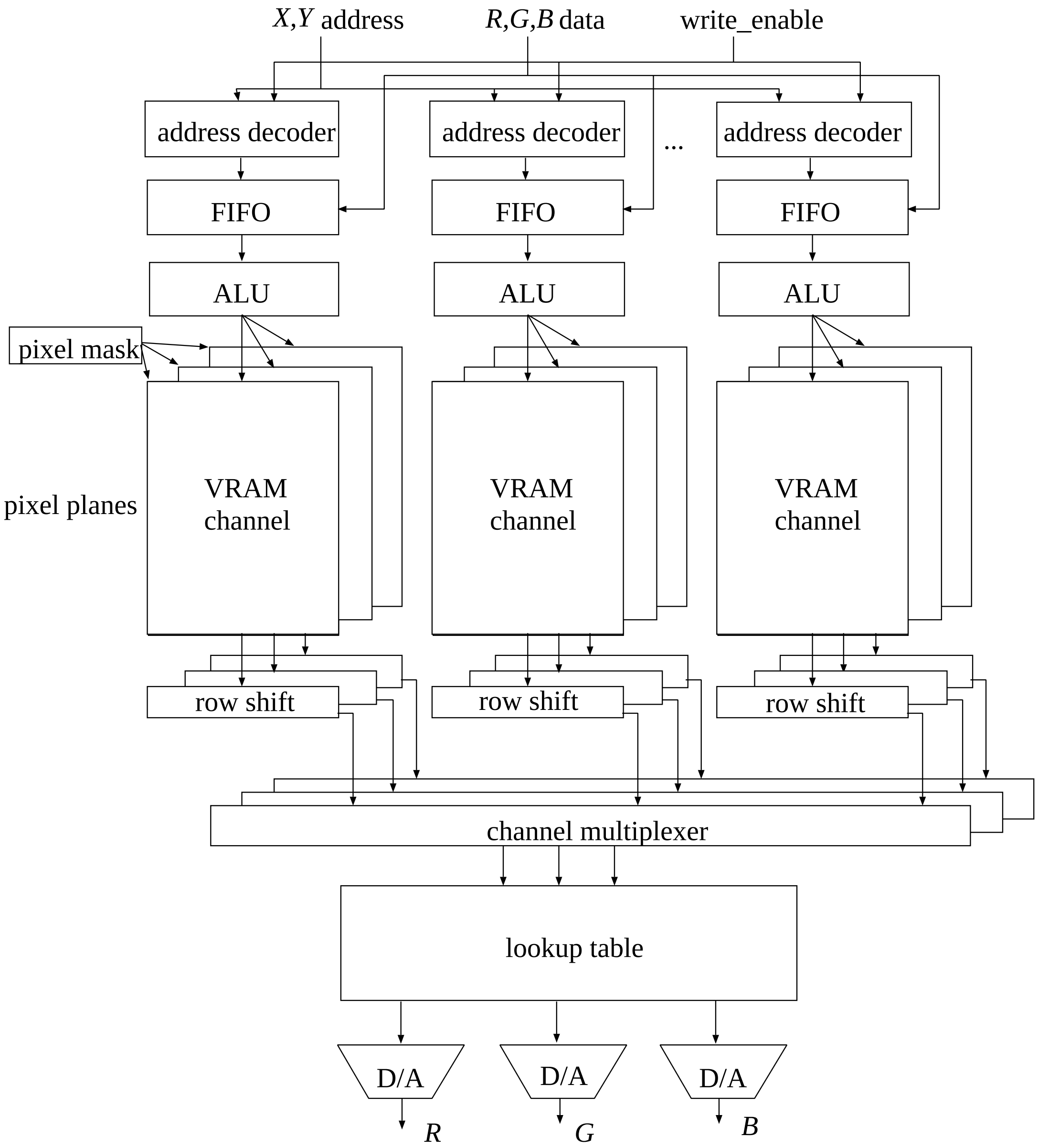

Figure 8.7: Frame buffer architecture

The problem of high speed drawing accesses can be addressed by partitioning the memory into independent channels and adding high-speed temporary registers or FIFOs to these channels. The write operation of these FIFOs needs very little time, and having written the new data into it, a separate control logic loads the data into the frame buffer memory at the speed allowed by the memory chips. If a channel is not accessed very often, then the effective access speed will be the speed of accessing the temporary register of FIFO, but if the pixels of a single channel are accessed repeatedly, then the access time will degrade to that of the memory chips. That is why adjacent pixels are assigned to different channels, because this decreases the probability of repeated accesses for normal drawing algorithms. FIFOs can compensate for the uneven load of different channels up to their capacity.

In addition to these, the frame buffer is also expected to execute arithmetic and logic operations on the new and the stored data before modifying its content. This can be done without significant performance sacrifice if the different channels are given independent ALUs, usually integrated with the FIFOs.

The resulting frame buffer architecture is shown in figure 8.7.