Mohammad Yunis (postgraduate student)
Seryozhenko A.A.
Donetsk national technical university, Donetsk
Faculty of computer science and technology
Department of computer engineering

SIMULATION OF GRAPHICAL PROCESSOR USING HDL

Until the early 1980s, computer graphics was a small, specialized field, largely because the hardware was expensive. The typical application was the image generators for trainer development and simulators. These systems must generate high quality images in real time. The concept of a "desktop" now became a popular metaphor for organizing screen space. Modern small personal devices, such as Nokia N810 Internet Tablet and Nokia M810 WiMAX Edition, are in want of real-time computer graphics which concerned with animation, video processing, 2D and 3D performing. Computer graphics systems' developing, as well as alter-idem, typically is performed years in advance of subsystem development and integration. In this process, models of functions and possible solutions for the physical architecture must be defined and matched to evaluate quality and select the most effective algorithm and the best possible hardware platform. For small personal systems designers the primary architectural/design issue the partitioning of system functionality across both hardware and software. Separate specifications for hardware and software, often written in non-formal languages, are delivered with functionality a priori, because changes to the partition may necessitate extensive redesign. Because software rework is viewed as easier than hardware redesign often drawbacks' corrections have a heavy software decision.

Real-Time Image Generator can be represented as a rendering pipeline - a logical model for computations needed in a raster-display system. It consists of a scene processor, geometrical, rastrization and video processors. It is not necessarily a physical mode, since the stages of the pipeline can be implemented in either software or hardware.

Functional organization of Scene Processor

Fig. 1 shows a version of the scene processor for the systems which use conventional primitives (lines and polygons) and priority algorithms with several levels of detail [2]. In accordance with the extended algorithm the scene manager includes:

- a matrix formation unit (MFU);
- a visibility detection unit (VDU);
- a detail/loader unit (DLU).

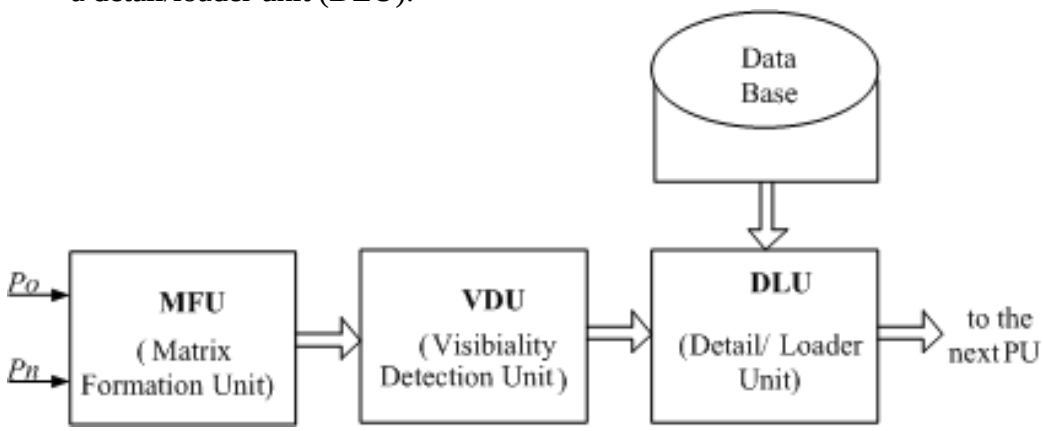

Figure 1 - Structure of Scene Processor

The first scene manager action begins with loading the vectors *Po* and *Pn* of the positions and orientations of an observer and an object.

Structure of positional vector

$$P = \{x, y, z, \psi, \theta, \gamma\},$$

where $\{x, y, z\}$ - a linear coordinate of the center of object/observer coordinate system;

$\{\psi, \theta, \gamma\}$ - an angle coordinate.

Using angels as table-pointers MFU takes *sin* and *cos* from the ROM and stores them it registers and accumulates matrix A and B coefficients [1].

Matrix A is used for translation from the object system to global. For translation from the global system to observer matrix B is used. To get A and B MFU takes $\{\psi, \theta, \gamma\}$ from *Po* and *Pn*, accordingly.

To translate center-point of object from the global system to observer multiplication to matrix is used [1].

The VDU sets a "visual" flag for potentially visible objects using its 3D spherical extents. The centers of the objects are transformed into the observer coordinate system for preliminary processing and setting "visual" flags.

To simplify the process the object's extent is analyzed, as bounded sphere with radius R_o. It is obvious; the object is located beyond the scope of visible region, if it is located behind the screen or outside the bounders of the simple viewing pyramid.

Algorithm of visibility detection is shown on fig.2. Sides of the base of visible pyramid:

$$A_w = \frac{a_w \cdot x_{con}}{d_w}; \quad B_w = \frac{b_w \cdot x_{con}}{d_w}.$$

where a_w, b_w - size of window;

d_w - distance from the observer to the window;

A_w, B_w - sides of the base of visible pyramid;

x_{con} - x-coordinate of the object center in observer coordinate system.

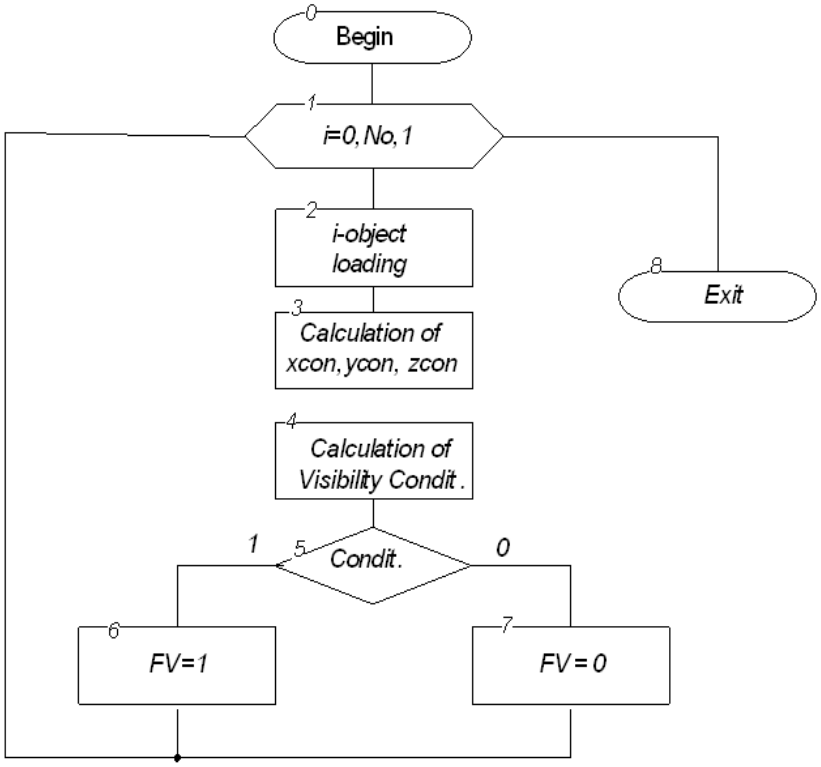

Figure 2 - Algorithm of visibility detection

Visibility condition:

$$\text{VisCond} = (x_{con} > d_w - R_o) \& (y_{con} < \frac{B_w}{2} + R_o) \& (y_{con} > -\frac{B_w}{2} - R_o) \&$$

$$(z_{con} < \frac{A_w}{2} + R_o) \& (z_{con} > -\frac{A_w}{2} - R_o),$$

where y_{con}, z_{con} - y-coordinate and z-coordinate of the object center in observer coordinate system.

The DLU forms the priority subobjects list for the current level of details. Expression for block 2 is:

$$S = (P_o.x - P_n.x) \cdot N_i.x + (P_o.y - P_n.y) \cdot N_i.y + (P_o.z - P_n.z) \cdot N_i.z, \qquad (1)$$

where $N_i = \{x, y, z\}$ - normal vector for *i*-node of BSP-tree.

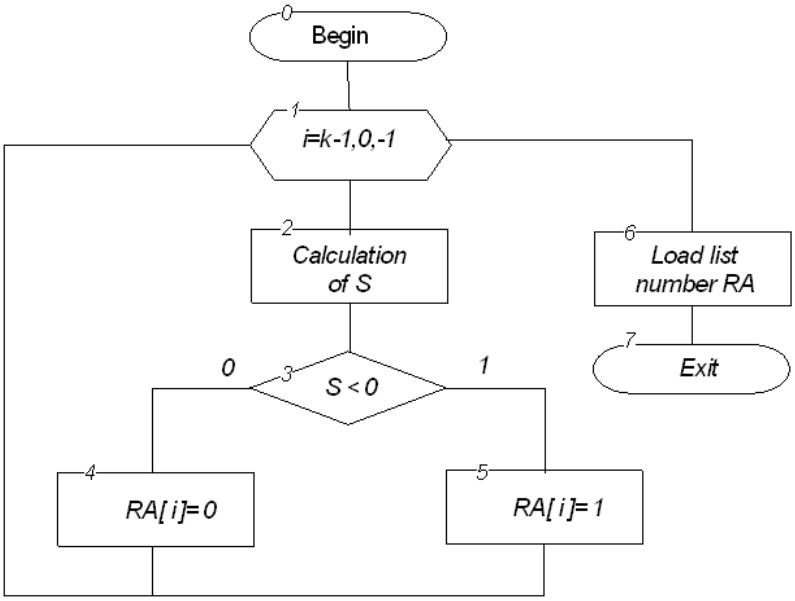

Figure 3 - Algorithm of priority-list loading

Realization of scene processor

Fig.4 shows an example of topological tree for Priority Algorithm. SP – parameters of normal vector of subdivision plane, "<" and ">" – sign of scalar product (expression 1).

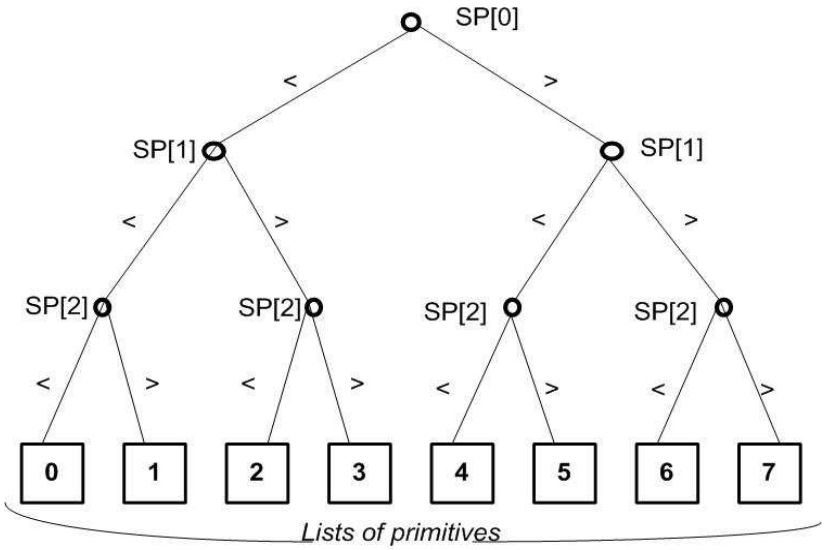

Figure 4 - Example of topological tree for Priority Algorithm

If scalar product negative, corresponding bit in number if list is cleared, if positive – set to 1. After calculations of all scalar products number of list is ready. Fig.5 shows an algorithm.

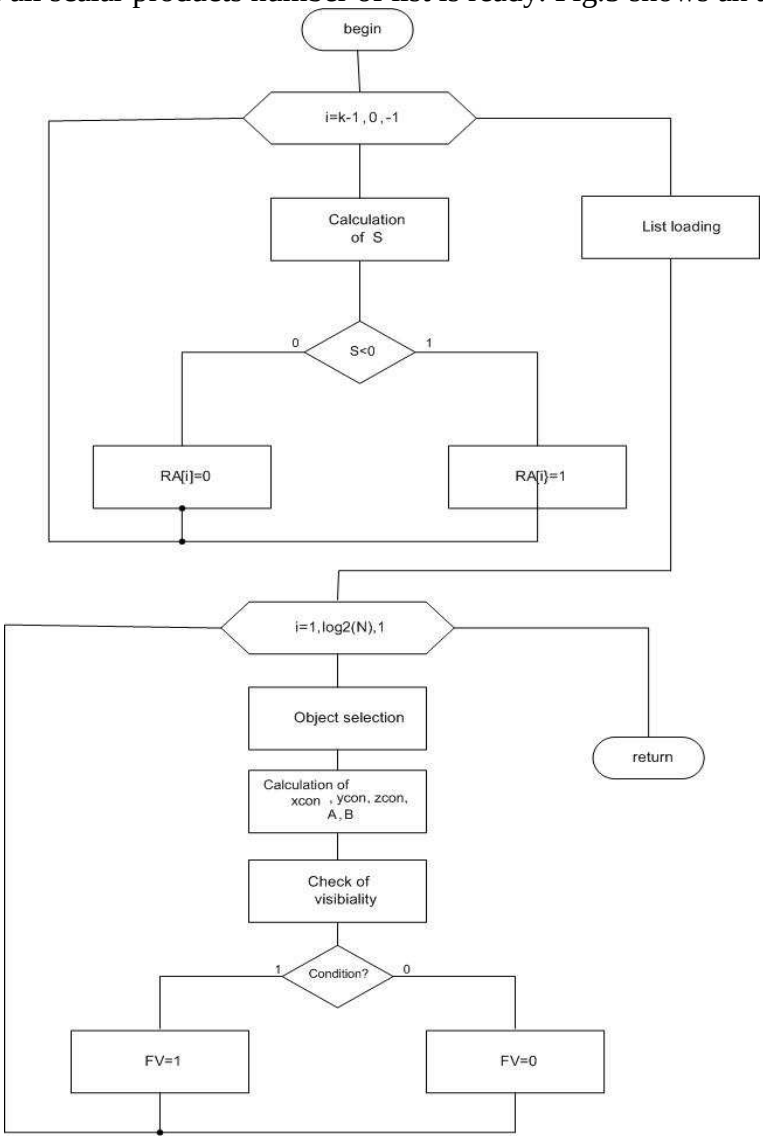

Figure 5 – Algorithm of local data base loading

Simulation on HDL

Direct transforming of UML state diagram into HDL is the first step in the automated synthesis of an FPGA circuits. UML has emerged as a common foundation for model driven architecture modeling. UML allows to build platform independent descriptions that can be used by designers to make informed decision about their hardware/software tradeoffs. UML is supported by a wide range of tools. The exchange of models between tools is supported by the XML standard, an XML-based description language which captures the details of UML model diagrams in a portable, machine readable format.

HDL is one of a class of computer languages used to provide formal description of electronic circuitry. An HDL standard text-based expression is capable of describing the temporal behavior and/or (spatial) circuit structure of an electronic system. HDL is widely used in hardware design to specify details of chip design for either specialized chips or FPGAs. For custom or standard-cell based integrated circuit, such as a processor or other kind of specialized digital logic chip, HDL specifies a model for the expected behavior of a circuit before that circuit is designed and built. Special logic synthesis tools are then invoked that ultimately provide the geometric information used to produce photolithographic masks necessary for the fabrication of the device.

For programmable logic devices such as FPGAs, HDL code is first delivered to a logic compiler (FPGA synthesis tool), and the output is uploaded into the device. The unique property of

this process and of programmable logic in general, is that it is possible to alter the HDL code many times, compile it, and upload into the same device for testing [3].

The transformation from high-level UML state diagram to HDL is based on a multi-step process, which consists of the following steps.

Step 1. State diagrams are created using UML state diagram notation. These diagrams describe system behavior using states, events and actions and correspond closely with the high-level design approach taken by circuit designers.

Step 2. UML diagrams are exported to XMI, a standard XML-based intermediate form. XMI uses predefined XML elements and attributes to specify the states, events and actions that make up the state diagram. The initial impetus for XMI was to enable UML diagrams to be imported and exported across different UML tools. Our approach uses the XMI as input to the next stage of processing.

Step 3. The XML representation of state machines is parsed by a Java-based XML parsing utility.

Step 4. Data extracted from the XMI by the Java parser is mapped to HDL templates, resulting in HDL suitable for use in FPGA construction.

As a result simulation program is developed. Fig.6 shows results of simulation.

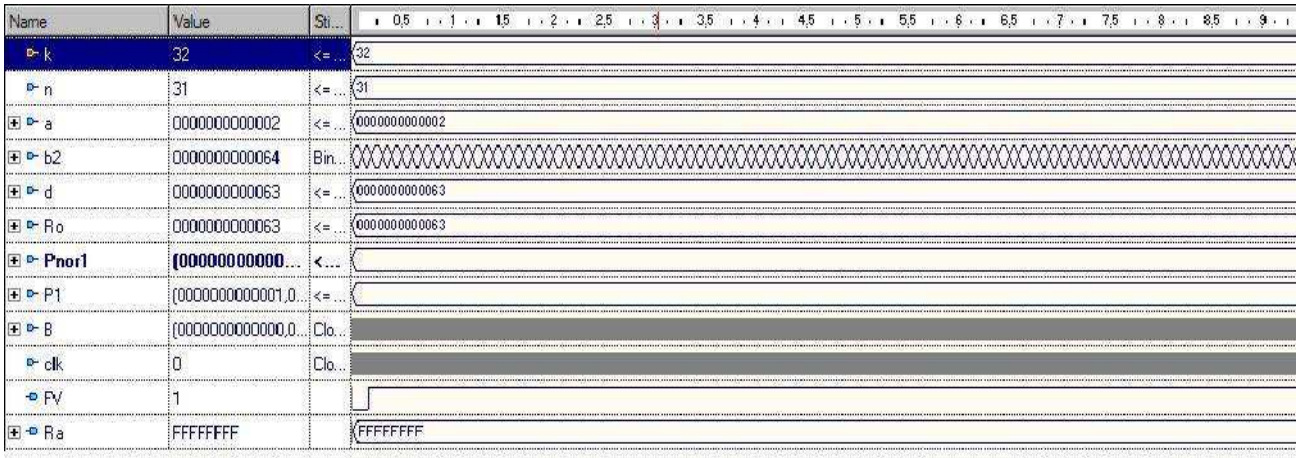

Figure 6 – Results of simulation

Summary

The ability to move from UML diagrams to HDL hardware descriptors is the first step in an effort to use model-based architecture to further optimize and automate the small systems development. Having made the decision to implement an algorithm using FPGAs, there are numerous decisions concerning the positioning of components that will impact product viability.

By analyzing the whole system in terms of supporting software and hardware, it is possible exploit new opportunities for image generation systems developing.

References

1. James D. Foley et al. Computer graphics: principal and practice. Addison-Wesley, 1997. – 1100 pp.

2 Malcheva R. The problems of modeling and rendering of the realistic complex scenes. Balkema: Proceedings of ECCPM, 2002. - PP. 537-538.

3. S.J. Mellor and M.J. Balcer. Executable UML. A Foundation for Model-Driven Architecture. Indianapolis: Addison-Wesley, 2002. – 322 p.