UDC 004.921

V.V. Krasnoproshin, D.I. Mazovka
Belarusian State University, Minsk
krasnoproshin@bsu.by, mazovka@bk.ru

Graphics Pipeline Extension Using Visualization Algebra

This paper describes one of the possible ways to the graphics pipeline automation. Introduced approach is based on specialized formal language called visualization algebra. We argue that proposed methodology can lower costs for target software development and build the way for further visualization automation.

Introduction

Most of the practical problems which are solved using computers require visual representation for the resulting data. As visualization is the easiest way for people to deal with complex information. That is why a scientific area of computer graphics became so popular. During its rapid development, computer graphics has formulated a plenty of sophisticated concepts, and one of the most important is a graphics pipeline. The notion of graphics pipeline stands for a set of approaches, methods, devices and software which implement the process of visualizetion.

The development of the graphics pipeline has been going so far in direction of complete automation. The principal part of its implementation consists of hardware, and software part plays mostly auxiliary interface role. This software is represented now by a number of hardware independent programming libraries [1]. At the same time, however, there are a lot of other problems not covered by the pipeline which stay between automated layer and the original application task.

This paper describes a potential approach for further graphics pipeline functionality extension and automation. This is achieved by introduction of a new visualization process abstraction level.

The aim of the research is to make a unification methodology for development of visualization systems.

Basic definitions

Below we describe main concepts and principles in computer graphics that help to understand the scope and placement of the problem under investigation. And the very core notion here is the computer graphics itself.

Computer graphics is an area in science and engineering where computers are used for images synthesization (creation) or processing of visual information which comes from the real world. In its other meaning, computer graphics refers to the product of such activity.

Visualization, in broad sense, is a process of data representation in visual form. The term may also be used for the result of such representation. In computer graphics visualization has a special definition, which is “rendering”. In this context visualization (or rendering) includes a set of hardware and software components which participate in the process of image generation. We can emphasize two basic methodological approaches to the rendering process implementation: rasterization and global illumination techniques (ray tracing, radiosity, ambient occlusion).

Rasterization is a rendering technique that implements transformation of vector data into plain two-dimensional images. This technique is a rough approximation of real processes, which are imitated in this case. However, computational performance is very high.

Global illumination techniques are based on modeling of real light transport processes using more or less accurate physical models. The resulting image quality is incomparably higher, but requires substantial computational powers.

Dynamic visualization used to employ only rasterization methods, because of the relative simplicity. But now we can see global illumination techniques coming into the stage. However, most of the current visualization hardware is based on rasterization rendering model.

Graphics pipeline

The process of rasterization in rendering is represented by the graphics pipeline. The pipeline technology was elaborated in the middle 90-th. In that time first video acceleration hardware, which automated some of the rasterization procedures, was created. Nowadays graphics pipeline includes many constantly improving components. In general, the pipeline can be visualized using the following scheme [2] (Fig. 1):

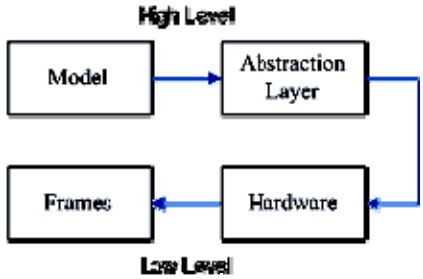

Figure1 – General graphics pipeline representation

The pipeline consists of two levels of automation: high and low. Low level includes specialized visualization hardware, and high level is programmable visualization software. These levels are split into four stages:

1. modeling, that means creation and updating of the visualized model;
2. hardware abstraction software level (*AL*), that means universal interface to the hardware;
3. graphics hardware (*HW*), that means low level visual processing;
4. frames output to the target video device.

The problem of performance is solved differently at different pipeline levels. At the low level, this problem is solved by increasing a number of processing units, volume and quality of memory. At the high level formalization of the visualization process is weak and this introduces some specific issues. We argue that standardization is one of the ways of improving the high level.

We offer an approach based on so-called visualization algebra, that is formal language for visualization description.

Visualization algebra

Rendering process model can be described with expression:

$$Model \rightarrow Frames \tag{1}$$

And with graphics pipeline stages included (Fig. 1):

$$Model \rightarrow \overset{A}{<} AL >\!\!\rightarrow\!\!< HW >\rightarrow Frames \tag{2}$$

Process *A* which conducts model transfer to the level *AL* is of our main interest. The process accepts an undefined computer model as an input and produces standardized sequence of data and instructions. The process implementation is rather difficult task, as data formats on input and output differ significantly. This leads to a necessity of creation of large sophisticated systems.

Natural solution in this situation is a creation of a middle data transformation level with splitting the process *A* onto two stages:

$$Model \xrightarrow{A_1} Visualization Model \xrightarrow{A_2} < AL > \rightarrow < HW > \rightarrow Frames \quad (3)$$

Here we have process A_1 (translation of initial model into formal representation *Visualization Model*), and A_2 (translation of this formal representation into standardized output for *AL*). Such approach gives us two benefits:

1. A_1 implementation is easier than *A* (less program code to write);
2. A_2 realizes translation from one formal representation to another, thus can be automated.

Currently the most widespread approach to implementation of the process *A* is scene graph [3]. In this approach visualized data is represented in a hierarchical structure, which includes grouping nodes and leaves. Data is taken from a set of predefined types: geometry objects, light sources, materials. Fig.2 shows an example of scene graph realization.

In SceniX graphics engine from nVidia [3] operations on scene data is performed with the help of traversal objects. These objects move through the tree and transform data into *AL* instructions sequence.

In general the traditional methodology of visualization process construction includes the following steps:

1. describe visualization task in standard terminology (geometry object, light sources, materials). If this is impossible, the task or the visualization systems should be modified;
2. build structured data representation, i.e. scene graph;
3. implement traversal operations that will translate scene data;
4. build traversal operation sequence.

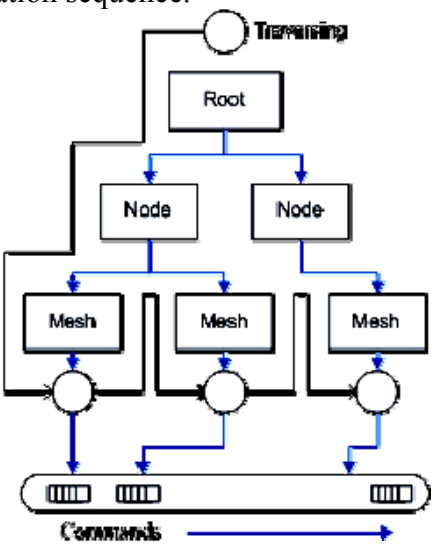

Figure 2 – Scene graph example

As though the traditional approach has proven its usefulness, it doesn't split the process *A* on independent parts well enough. Its flaws show up when we try to build untypical visualization processes. In this case developer either has to reformulate the initial requirements or tweak the visualization system itself. For example, the task of visualization of 2D interfaces

can hardly be described in scene graph terminology and often is represented by separate components in traditional systems. Another drawback is that traversal operations are multifunctional (like in SceniX example). This leads to low algorithmic specialization and violates modular code structure.

Thus scene graph approach can be considered as incomplete, and we still need a way to lower complexity of the process A. To solve this problem we will study the rendering process more precisely.

The model of visualization task can be represented in this form [4, 5]:

$$VisualizationModel =< Data, PU > \quad (4)$$

Where *Data* is a set of visualized objects. *Processing Units* (*PU*) are procedures that transform data during A_2 execution. In this way the process A_2 translates model data into instruction sequence for *AL*. This process is shown in Fig. 3, again we have a hierarchical structure, but here nodes are represented by *Processing Units*.

As input the process accepts Scene object is a set of visualized model objects, and produces a sequence of instructions as output.

In this paper we don't consider implementation details of A_2 process, but we study the first part of the problem and particularly the process A_1.

To describe the process A_1 we introduce the following set of objects:

1. scene (a set of visualized model objects: $S = \{O_i\}$);
2. scene object (an independent scene entity which is described by a set of attributes: $O = \{A_i\}$);
3. scene object attribute (data chunk with of particular type);
4. frame (a result of data processing on the graphics pipeline, two-dimensional vector array. Can be used as an attribute.)

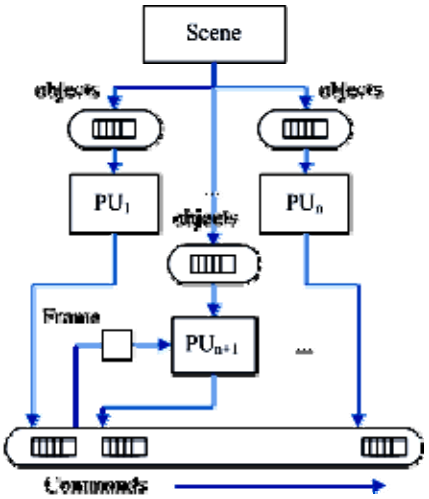

Figure 3 – A_2 example

These objects generalize a set of all possible objects of the visualization task, including those of traditional approach. From the functional point of view, we introduce a set of operations that includes:

1. sampling (separation of a group of objects from a larger set using some particular condition):

$$Sample(A) = B, B \subseteq A;$$

2. object shader (transformation of object group into one or more frames):

$$ObjectShader(\{O_i\}) = \{Frame_j\};$$

3. frame operators (transformation):

$$FrameOperator(\{Frame_i\}) = \{Frame_j\}.$$

These operators generalize and categorize existing methodology as well as objects above.

The formal language of description of *Visualization Model* with introduced objects and operators we will define as visualization algebra. Algebraic system in our definition is a set of objects, operators and rules of their application.

Visualization algebra is the main tool in translation of computer model *Model* to *Visualization Model* representation. Rendering process description, using visualization algebra, corresponds the equation:

$$Foo(S) = F \quad (5)$$

Here scene S is transformed into resulting frame F by process *Foo*, which is constructed with the algebra operators. The process A_1 in this case is a process of construction of operation *Foo*.

Creation of a visualization system requires two steps: at first we need to define all the visualization algebra operators necessary, and then describe visualized objects characteristics. If some visualization system allows defining any number of various objects and operators via its interface, we will call such system functionally consistent.

If system doesn't allow including objects of any kind, we can show existence of computer model that it wouldn't be possible to reflect onto the system classification. If the system is unable to support various visualization operators, then it cannot be used to describe the rendering process in form (5) by design.

The following sequence describes the process of visualization task solution using visualization algebra:

1. initial model data adaptation to algebra data entities;
2. implementation of necessary operators;
3. construction of visualization process equation.

Thus, while traditional approach concentrates attention to data representation methods, visualization algebra emphasizes the process of visualization itself. This introduces a new abstraction and formalization level.

The proposed approach is rather universal, as literally all of the current visualization systems employ the same methods and algorithms, and the difference comes mostly in data representation formats. Usage of visualization algebra helps to separate and automate common processes. And standardized visualization task format eases development and integration of such systems. In the next section this difference will be illustrated.

Application example

To illustrate key points of the visualization algebra approach we will look at the task of city planning. Infrastructural and architectural problems cannot be solved here without proper visualization. So visualization task in general would require rendering of architectural data provided by city developers. Given no more details at the beginning we will compare projects' life cycles.

General approach

Input: architectural data (scene) in some 3D graphics description format.

1. The system may or may not support the format. If data format is not supported, input data should be converted.
2. The system renders the scene in default mode.

Visualization algebra

Input: architectural data (scene) in some 3D graphics description format.

1. The data should be converted into standard visualization format.
2. Using standard visualization methods we construct rendering equation:

```
extern Scene scene;
extern Sampler camera;
Frame main()
{
  return StandardObjectShader(camera(scene));
}
```

Here scene and camera objects are input parameters for the pipeline, and StandardObjectShader is a standard rendering technique.

At the next stage, another requirement is introduced into the visualization task: dynamic scene lighting. I.e. rendering should support changing illumination environment.

General approach

1. The system may or may not support the visualization effect. If the effect is not supported, it should be implemented using SDK of the system.
2. The system renders the scene.

vis ualization algebra

1. Rendering equation in this case changes to:

```
extern Scene scene;
extern Sampler camera;
Frame main()
{
  Sample visible_objects = camera(scene);
  Sample lights = LightsIntersectObjects(scene, visible_objects);
  return DiffuseObjectShader(visible_objects) *
    LightObjectShader(lights, visible_objects);
}
```

Here we sample visible objects and then all the lights which intersect them. DiffuseObject-Shader then renders the objects with no shading and the result is multiplied by the result produced by LightObjectShader which renders light effect.

If any of the object shaders is unavailable, it should be implemented.

If the task requires more substantial changes, general approach may fail to meet them, as SDK doesn't guarantee applicability for the changes necessary. In this case the user may need to tweak the system to implement missing features, or change the system itself. Both variants are very time consuming.

In case of visualization algebra we have common interface for any use case. And uncommon requirements won't differ anyhow from standard ones. Visualization algebra also promotes visualization system agnostic view on the rendering pipeline. This has much common with HAL.

Conclusion

In this paper we studied the problem of organization of real-time visualization systems. We proposed an approach to formalization of visualization processes using visualization algebra. This method lowers implementation costs and is a convenient software interface for development of visualization systems, which is in similar by the idea to relational algebra in databases.

Literatura

1. Microsoft DirectX documentation (August 2009).
2. Akenine-Möller T. Real-time rendering / T. Akenine-Möller, E. Haines, N. Hoffman. – [3rd ed.] Wellesley, Massachusetts, 2008.
3. NVIDIA® SceniX™, http://developer.nvidia.com/object/scenix-home.html
4. Mazouka D. Effective approach to the computer models visualization problem. Proceedings of the conference / D. Mazouka // Pattern Recognition and Information Processing (PRIP'2009, Minsk, Belarus 19-21 May 2009).
5. Krasnoproshin V. The problem of abstract model visualization / V. Krasnoproshin, D. Mazouka // Vestnik BSU. – № 2 series.1 (2010). – P. 130-134.

В.В. Краснопрошин, Д.И. Мазовка

Расширение графического конвейера с использованием алгебры визуализации

В статье описывается один из возможных путей автоматизации графического конвейера. Представленный подход основывается на специализированном формальном языке, называемом «алгебра визуализции». Предложенная методология, по мнению авторов, снижает стоимость разработки соответствующего программного обеспечения и устанавливает предпосылки для дальнейшей автоматизации процесса визуализации.

В.В. Краснопрошин, Д.І. Мазовка

Розширення графічного конвеєра з використанням алгебри візуалізації

У статті описується одна з можливих доріг автоматизації графічного конвеєра. Представлений підхід ґрунтується на спеціалізованій формальній мові, званій «алгебра візуалізації». Запропонована методологія, на думку авторів, знижує вартість розробки відповідного програмного забезпечення і встановлює передумови для подальшої автоматизації процесу візуалізації.

Статья поступила в редакцию 16.06.2011.