The RTSS Image Generation System

K. Alvermann and **S. Graeber**
Deutsche Forschungsanstalt für Luft- und Raumfahrt
Institute of Flight Mechanics
Lilienthalplatz 7
38108 Braunschweig
Germany
E-Mail: alvermann, graeber@fm.bs.dlr.de

J.W.L.J. Mager and **M.H. Smit**
TNO Physics and Electronics Laboratory
Oude Waalsdorperweg 63
2509 JG The Hague
The Netherlands
E-Mail: Mager, Smit@fel.tno.nl

1. SUMMARY

Main market demands for the visual system of a simulator are photorealism and low latency time. RTSS, a general purpose image generation module developed within the European ESPRIT project HAMLET, can meet these demands through the use of High Performance Computing technology. This technology provides the needed communication and computing power. Moreover, by using parallel processing, the whole system is scalable, i.e., the same software and hardware design can be used for small, cheap systems, as well as for high-end view simulations. This allows an easy adaptation to the user's needs.

RTSS also includes an object and scenario editor implemented on a work station, as well as filters to other object data standards.

This paper will give an introduction to the soft- and hardware design of RTSS. It will then present the features of the system as well as the interfaces: the filters to import external model data and the interfaces to the simulation system itself.

2. RTSS

2.1 Overview

Within the context of the European ESPRIT Project 6290 HAMLET, TNO Physics and Electronics Laboratory (TNO-FEL, NL), Deutsche Forschungsanstalt für Luft- und Raumfahrt (DLR, D), and Constructiones Aeronauticas (CASA, E) developed the Real-Time Simulation System (RTSS).

The work in the HAMLET project is done within the framework of the ESPRIT program and partly funded by the Commission of the European Communities. The following companies form the HAMLET consortium: AEG (D), CAP Gemini (F), CASA (E), DLR (D), HITEC (GR), INESC (P), INMOS (GB), Parsytec (D), Gabriel (GR), TNO (NL), and TU Munich (D).

The RTSS is a general purpose image generation module. RTSS can be used by simulators, e.g., flight simulators (see Fig. 1), and other man-in-the-loop applications which require visual feedback.

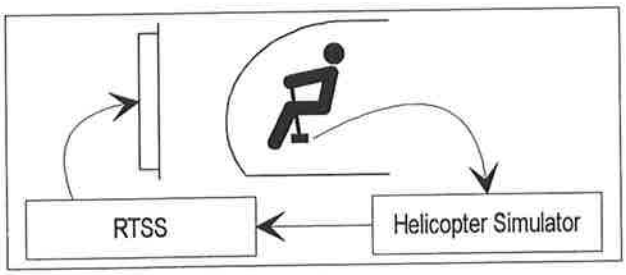

Figure 1: The RTSS Concept

RTSS has to generate photo-realistic images in real-time. To realize this, the underlying hardware has to provide a high computation power. Moreover, the data bandwidth needed to communicate the large data flows processed by the RTSS places severe demands on the communication power of the underlying hardware.

RTSS is designed to be scalable. If the demands of an application are rather low, a small (and, therefore, cheaper) system can be used. If the demands get higher, the RTSS can be expanded to fulfil the growing requirements. Requirements are the image rate and the latency time, as well as features like shadow generation, anti-aliasing, and the size of the database.

RTSS is divided into an off-line and an on-line part. The off-line part consists of an interactive object and scenario editor and filters to input data from known geometric formats such as the MultiGen Flight format. The objects and scenarios are stored in a database in a file system.

The on-line part inputs this database, i.e., data describing the geometry and appearance of objects, light parameters, image parameters, etc. RTSS then outputs the corresponding image. As a second task, RTSS contains a collision detection module. This module checks the moving objects for collisions and reports them back to the application. During run-time, the application can change almost all parameters, e.g., the position and orientation of all objects and of the camera, light positions and parameters, etc.

The interface to the application is a bi-directional channel. The application sends commands (parameters, movement data, object changes, etc.) to the RTSS and receives in turn the results of the collision detection.

Paper presented at the AGARD FVP Symposium on "Flight Simulation — Where are the Challenges?", held in Braunschweig, Germany, from 22-25 May 1995, and published in CP-577.

2.2 Requirements and Architecture

Using RTSS in man-in-the-loop applications leads to the following main requirements

- real-time generation of images,
- an ergonomic image rate,
- a low latency time, and
- high resolution, photo-realistic images.

Since no restrictions in the movement of objects are acceptable, the images cannot be pre-calculated but must be calculated in real-time. To guarantee a smooth movement of objects in the image, the image rate must be about 20–30 images per second (depending on the dynamics of the application). A low latency time is dictated when RTSS is used in man-in-the-loop applications. The time between the command of a user in the application and the display of the corresponding image, i.e., the latency time, should be below 100 milliseconds to prevent motion sickness or severe timing differences between the simulator and the real thing. The image must be realistic enough to serve as a visual orientation. It should contain shadows as a visual clue for the position of objects, a high resolution, and anti-aliasing techniques should be used to suppress edged lines and edged object boundaries.

The above requirements led to the following specifications of the RTSS. For the off-line definition and construction of objects, RTSS provides:

- an interactive, three dimensional scenario and object editor;
- import of external model data (MultiGen, AutoCAD, etc.);
- static and dynamic objects with several levels of detail;
- hierarchic grouping of objects;
- objects are built up from points, lines, and planar polygons;
- polygons can be textured;
- multiple dynamic directional light sources.

During run time RTSS reports:

- collisions between objects on polygon level.

The following features are used for the resulting image:

- light reflection and emission using multiple light sources and several lighting models;
- shadow generation;
- anti-aliasing and depth cueing (fog simulation).

The performance specifications are:

- true colour (i.e., 24bit) images at high resolutions;
- 25 images per second;
- a latency time of 80 milliseconds.

2.3 Architecture

RTSS is decomposed into three subsystems, corresponding roughly to the three companies involved in its development: the Scenario Creation (CASA), the Simulation Execution (DLR), and the Image Generation (TNO) subsystem (see Fig. 2.)

The Scenario Creation subsystem (see chapter 3) is the off-line part of the RTSS and includes the scenario editor, the object editor, and the external model data import module. The resulting scenarios are stored in files to be loaded by the on-line part of the RTSS.

The Simulation Execution subsystem (see chapter 4) loads the scenario, handles the communication with the application and the database. It converts the object data into primitives (points, lines, faces, and shadow faces) which are sent to the Image Generation subsystem.

The Image Generation subsystem (see chapter 5) produces the image from the primitives supplied by Simulation Execution. Textures are mapped onto the appropriate polygons, the polygons are scanned line by line, and the lines are rendered into an image buffer. Shadows are added and anti-aliasing techniques are used for the final image.

The on-line part of the RTSS is designed to be scalable. Computational intensive processes (e.g., database handling, scan conversion, texture mapping, shadow generation) are done in parallel. The number of parallel processes is variable and determines the power of the system.

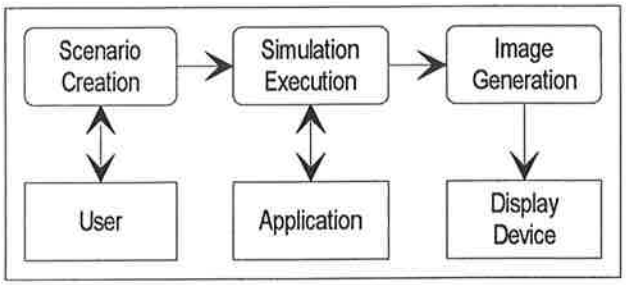

Figure 2: RTSS Architecture

3. SCENARIO CREATION

Before a simulation can be executed, the scenario of this simulation has to be defined using the scenario creation tools. These tools provide means to import object models from external modellers, to edit the characteristics of the objects contained therein, and to edit the scenario itself. Scenario creation is done off-line on a host computer.

3.1 Import Filters

The *import filters* provided in RTSS enable a user to import geometric object models created by existing commercial modellers. The imported model data is converted into the internal format used by RTSS keeping as much of the semantics of the model as possible. Currently, RTSS supports models made using MultiGen in the Flight format as well as AutoCAD.

3.2 Object Editor

The user can interactively modify the visual characteristics of an object using a built-in *object editor*. This editor consists of two parts: a material editor and a texture editor. The material editor can be used to combine several characteristics of materials. A material defines transparency, lighting type (emitter or reflector), shading type (Flat, Gouraud, or Phong), and reflection coefficients for ambient, diffuse, and specular lighting. In the texture editor, the user can change the material of each surface of an object and apply textures to these surfaces.

3.3 Scenario Editor

Using *the scenario editor*, the user finally specifies the scenario contents. It contains the initial position of the elements (*assemblies*, *cameras*, and *light sources*) in the scene. An assembly is a hierarchically organized set of objects with fixed relative positions. For an assembly, the user can specify whether it is shadow casting and visible or not and its collision detection list. During simulation,

the assembly is tested for collision with all assemblies identified in this list. The user can define the type of a light source (ambient or directional) and the values corresponding to this type. For a camera, a user can define its aperture angle, view direction, and clipping planes.

Most of the properties of the elements can be changed during run time. Therefore, the scenario editor enables a user to specify whether during simulation changing the properties is allowed (element type is *dynamic*) or disallowed (element type is *static*). Apart from the properties of the elements, the scenario also contains global information, so-called *session parameters*. These include parameters for the lighting model, screen resolution, depth cueing, background colour, initial camera, and settings for switching on/off edge anti-aliasing, texture anti-aliasing and shadow generation. The depth cueing parameters as well as the background colour can be changed during run time. While defining the scenario, the user can inspect the scene using a 3-D viewer. This shows the scene from a global view point or from the view point of the initial camera. In this scene icons indicate the position of light points and cameras.

4. SIMULATION EXECUTION

The Simulation Execution subsystem is responsible for the interface to the application and the handling of the database. The individual modules of the subsystem are shown in Fig. 3.

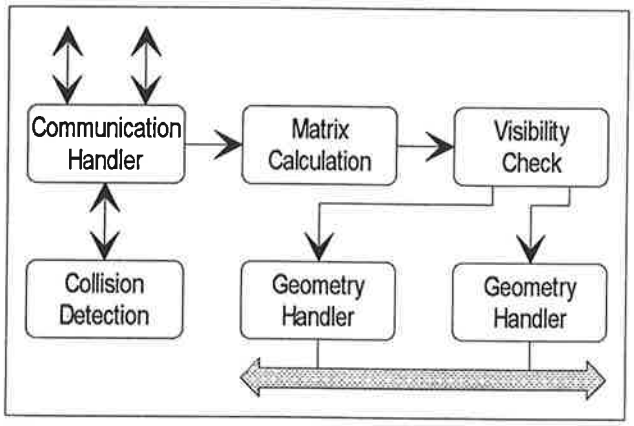

Figure 3: Structure of Simulation Execution

4.1 Interface to Application

The interface to the application is a bi-directional channel. At the begin of a run, the application sends the name of the scenario to be used, which is then loaded from files (thus, not through the application). During run time, the application sends commands. Commands are:

- parameter changes: lights, background colour, depth cueing;
- camera changes: position, orientation, parameters;
- movement data: position, orientation, and scale of objects;
- object data: colour and material of objects;
- collision detection: which object pairs are checked for collisions.

Commands have to be issued in a specific protocol. There is no restriction on the timing of the commands, since the reception is decoupled from the rest of the system. Therefore, they may be sent at any time.

RTSS in turn sends back the result of the collision detection. Collision detection, and thus the feedback, can be switched off it it is not needed.

4.2 Collision Detection

The Collision Detection module is an optional part of the RTSS. It detects overlapping geometries of objects on polygon level, i.e., the module reports polygons of different objects that intersect each other. For each assembly it can be specified with which other assemblies collisions are checked. This can be used to reduce computation to the interesting objects (one space ship making contact with another) and to avoid unnecessary collision reports which will occur in any case (a car moving on the street). Collision Detection is decoupled from the rest of the system (for input as well as output), i.e., the reports to the application are not coupled to the frame rate.

4.3 Database

The whole database is distributed over a number of processors, the *Geometry Handlers*. During a frame the *Visibility Check* issues object identifiers of visible objects and shadow casting objects to the respective Geometry Handler. Objects not visible in the image or not casting a visible shadow are discarded as early as possible. The Geometry Handler converts the object into primitives (points, lines, faces, and shadow faces), transforms these primitives to the correct coordinate system, and clips them against the view volume. The lighting model is evaluated for vertices and faces. For shadow casting objects shadow volumes are calculated and decomposed into shadow faces. All primitives are collected in a buffer and broadcasted over a bus to the Image Generation subsystem upon a system synchronization signal. For details of the algorithms see [1].

The number of the Geometry Handlers is a parameter of the system design. Systems using small databases need only one or two Geometry Handlers, while systems using big databases can use many. The number is not limited by software or hardware restrictions.

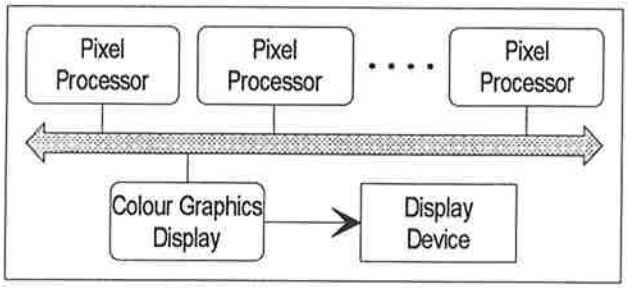

Figure 4: Structure of Image Generation

5. IMAGE GENERATION

5.1 Operating Principle

The output image of RTSS is divided into a number of horizontal *scan lines*, each of which is divided into a number of *pixels*, thus forming a rectangular raster. The Image Generation process is responsible for the determination of the colour of every individual pixel. This is based upon the *render primitives*, which are the output of the Simulation Execution subsystem (see chapter 4). First, all the render primitives are rendered in the *frame buffer*. Then all the shadow face descriptions are processed into a separate *shadow buffer*. Finally, the frame and shadow buffer are merged into the final *image buffer*. If there are no shadows present in the scenario, the last two stages are skipped and the frame buffer is considered to be the image buffer.

5.2 Parallel Processing

The computations involved in Image Generation are the most time consuming of RTSS. Therefore, an efficient way of decomposing this process into parallel tasks is required. The decomposition of Image Generation is shown in Fig. 4

The output image is divided across the processors executing Image Generation (these processors are called *Pixel Processors*) in a *scan line interleaved* fashion. Suppose the number of pixel processors to be N. Pixel processor i (with $i < N$) 'owns' scan line i and then $i + N$ and so on. This way, every pixel processor has roughly the same number of scan lines (and therefore pixels) to compute, thus guaranteeing a uniform workload.

The primitives are received via a bus to which all Pixel Processors, as well as the Geometry Handlers, are connected. After every Pixel Processor has finished, the partial images are gathered in the video memory of the Colour Graphics Display for display. This gathering is done across the same bus. The number of Pixel Processors can be adapted to the required performance of the system. It is not limited by software or hardware restrictions.

5.3 Algorithms

Visible priority of occluding primitives (also called Hidden Surface Elimination, HSE) is treated using the *z-buffer* algorithm. Almost all major visual systems use this (or adaptations of it) for HSE. The algorithm has the advantages that it is

- Independent of the order in which the primitives are treated. This is of particular importance in the RTSS case, because a global priority ordering of the primitives (which would be needed otherwise) can not be performed by the Geometry Handlers (when implemented on more than one processor). Therefore, this ordering would have to be executed by the Pixel Processors, which is are already heavily loaded with work.
- Capable of handling all sorts of render primitives.

To prevent disturbing noise in the image, two forms of anti-aliasing have been applied:

- Lines and the edges of polygons are smoothened using the *a-buffer* algorithm, described in [2]. This technique filters the well known 'staircase jaggies'. We have chosen this method from a number of alternatives, because it is suited for a z-buffer set-up, only uses extra computing power for filtering of the edges, is independent of the sub-pixel resolution used, and does not require a major amount of extra memory.
- A technique called *mipmapping* is used to filter the inside of *textures*. Almost all commercial visual systems which incorporate texture anti-aliasing use mipmapping. Refer to [3] for details on this technique.

6. HARDWARE

The demands for the on-line part in terms of calculation power, communication bandwidth, and scalability, are satisfied by using parallel processing. To guarantee a long life cycle and easy expandibility the hardware developed inside the HAMLET project was used. The processor is a PowerPC connected to a T425 transputer for communication. However, the bandwidth needed to transport the primitives and the image cannot be met by transputer links. Therefore, a special bus system called Transputer Image Processing (TIP) bus is used.

6.1 Processors

The processor board TPM-MPC is equipped with a PowerPC 601 processor running at 80 MHz. An INMOS T425 transputer running at 30 MHz is used for communication along the 4 transputer links operating at 20 Mbit/s. The transputer is equipped with 4 Mb of local memory and shares 16 Mb of memory with the PowerPC. Using the transputer links, these boards can be connected in any kind of network. The PowerPC and the transputer are programmed in C using the PowerTools which are an extension of the INMOS C toolset.

6.2 Communication

The high communication demands required by the communication of the primitives (see section 4.3) and the image (see section 5.2) cannot be met by using the transputer links (about 2 Mb/s): a 512×512 true colour image with 25 images/s needs a bandwidth of 25 Mb/s; the respective 1024×1024 images needs 100 Mb/s. Additionally, the primitives are broadcasted over the bus.

The demands are satisfied using the TIP-Bus. This bus provides the hardware to transfer data from the local memory of one processor to that of another. The bus is 32 bit wide and has a peak bandwidth of 120 Mb/s. The bus architecture allows point-to-point communication as well as data broadcasting and gathering (which is needed to distribute the primitives to all Pixel Processors and to gather the partial images on the Display Processor). If necessary, the bus can be divided into several segments working in parallel.

The TIP-MPC boards are equipped with a PowerPC 601 operating at 80 MHz, an INMOS T425 transputer running at 25 MHz as the bus controller, 16 Mb of shared memory, and 2 Mb of video memory, which is the interface to the bus. Any number of these boards can be connected to one bus.

Special interface boards are available to connect the TIP-Bus to display systems or cameras. RTSS uses the Colour Graphics Display. The CGD is equipped with an INMOS T805 transputer and a video chip to drive a display device.

The CGD is also programmed in C using the PowerTools. The TIP-Bus can either be programmed directly or using a special language called TIP-Set.

7. REFERENCES

1. Foley, J.D.; van Dam, A.; Feiner, S.K.; Hughes, J.F., "Computer Graphics", Addison-Wesley Publishing Company, Second Edition, 1990.
2. Carpenter, L., "The A-buffer, An Antialiased Hidden Surface Method", Computer Graphics Vol. 18, No. 3, July 1984 (SIGGRAPH 84).
3. Williams, L., "Pyramidal Parametrics", Computer Graphics Vol. 17, No. 3, July 1983 (SIGGRAPH 83).