JaGrLib: library for computer graphics education

Josef Pelikán
Charles University in Prague
Malostranské nám. 25
CZ-11801, Praha 1,
Czech republic
Josef.Pelikan@mff.cuni.cz

Jan Kostlivý
Charles University student
Malostranské nám. 25
CZ-11801, Praha 1,
Czech republic
kostlivy@centrum.cz

ABSTRACT

This article describes JaGrLib – environment for experimenting with algorithms, data structures, general approaches and implementation patterns. This framework was primarily developed to help teaching of computer graphics at Charles University in Prague. But it can be used as a handy tool for experimenting in any branch of applied computer science where modularity, versatility and reusability are key features. The system is implemented in Java language and takes advantage of its object-oriented design, good portability, flexibility, etc. At the end of 2003 a new graphical environment was developed and the whole library is publicly available with dozen of modules – mainly from field of computer graphics.

Keywords

JaGrLib, modular framework, computer graphics, education, Java.

1. INTRODUCTION

Back in 2000 we started to look for a new system of student assignments in computer graphics curicullum at Charles University. After short period of studying existing systems a decision was made to build a new framework in-house. The most important requirements and propositions were:

- fine-grain modularity of the system is crucial.
- code readability and clarity is more important than effectivity (with minor exceptions).
- versatility and flexibility of modules (leads to best reusability).
- even person with little knowledge of library internals must be able to contribute to it (write/adapt modules).
- module programmers should concentrate to the computer graphics problem (i.e. essence of it), not to technical details and tricks.
- portability to various computer platforms.

Finally we have decided to bind this new solution to Java language to achieve high level of flexibility and comprehensibility. Main building blocks (modules, pieces) have form of **Java classes**, connections between modules are controlled by **protocols** (called "interfaces" in Java). These ideas match well to principal rules of object-oriented design and programming.

Journal of WSCG, Vol.12, No.1-3, ISSN 1213-6972
WSCG'2004, February 2-6, 2003, Plzen, Czech Republic.

The **JaGrLib framework** was developed during last three years. Thanks to students of computer graphics at our university, many modules for 2D and 3D graphics are available today. Some of them are not coming with public source code yet, simply because they are going to be used as student assignments once more.

Jan Kostlivý is finishing its Bc. thesis by implementing graphical user interface for the library [GUI03]. His GUI system was designed to facilitate routine work with JaGrLib system.

Next sections describe basic JaGrLib concepts, show some details about GUI environment and give examples of more complicated compositions.

2. JAGRLIB CONCEPTS

"**Modules**" (descendants of the class "Piece") are basic building blocks of the framework. Each module has one or more "**plugs**" – exclusive interfaces from inside of a module to the rest of the world. Each Plug is tagged by an "**interface**" (see [JDK]), so this module can call another one via this interface or vice versa.

So called "**input plugs**" represent interfaces (protocols) which are implemented by this module – they can be used (called) from other modules. On the

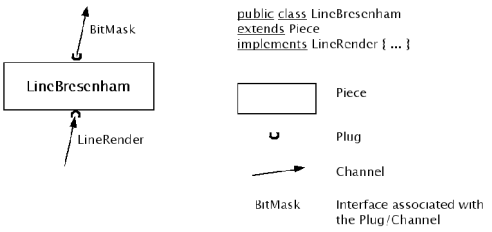

Figure 1. Key terms: module (Piece), plug, channel

other side, a plug marked as "**output**" represents need of some external functionality: another module with desired function should be connected here as a "black box".

Note that "input" and "output" attributes are not expressing any form of causality or direction of "data flow" – they only represent direction of method-calling in Java language ("clients" are calling methods of "servers").

Two plugs are declared as "**compatible**" if they are tagged with compatible interfaces and have opposite direction. Two interfaces are compatible if the output one is sub-interface of the input one (functions needed by a client have to form a sub-set of functions implemented by a server).

Compatible plugs can be connected together using "**channels**". Channels are actual connections between modules, they are established in running stage of a program. Channel is able to connect one input plug (on the side of implementor, server) with many compatible output plugs (callers, clients).

Special channel named "**multi-channel**" can act as "call-multiplier", one output channel is connected to several input plugs; each instance of function call have to be cloned for individual implementors. Of course this mechanism does not make much sense in context where return values of methods are important.

The whole assembly of modules connected together using channels is called "**composition**".

See Fig.2 for example of working composition (program for testing of one line-drawing algorithm).

3. GRAPHICAL USER INTERFACE

It is clear that much routine work must be made to prepare working composition well. In a process of experimenting with algorithms and data structures there is need for procedures for rapid manipulating with modules and compositions. Some examples are: choice of suitable module for required task (interface), replacing functional module with another – but compatible – one (due to comparison), grouping/ungrouping of modules to autonomous sub-compositions, etc.

GUI environment was developed by Bc. student Jan Kostlivý (see [GUI03]). It serves not only as user-friendly interface to JaGrLib core but also introduces some new and improved mechanisms for module management. GUI environment is implemented in Java language with use of universal graphical interface Swing. All data files are stored in XML format and are ready for future extensions.

Module registration

Every active module has to be registered in JaGrLib database. Module attributes are: **name** (name of Java class), **package**, implemented (exported) and needed (imported) **interfaces** in form of input and output **plugs**, text identification in three fields: "**name**", "**category**" and "**description**" and module type – "**template**" (see below).

GUI system includes effective mechanism for choosing modules from the registration database – user can restrict set of listed modules applying various filters (mostly used filtering attributes are: module name and category, set of interfaces/plugs, module template).

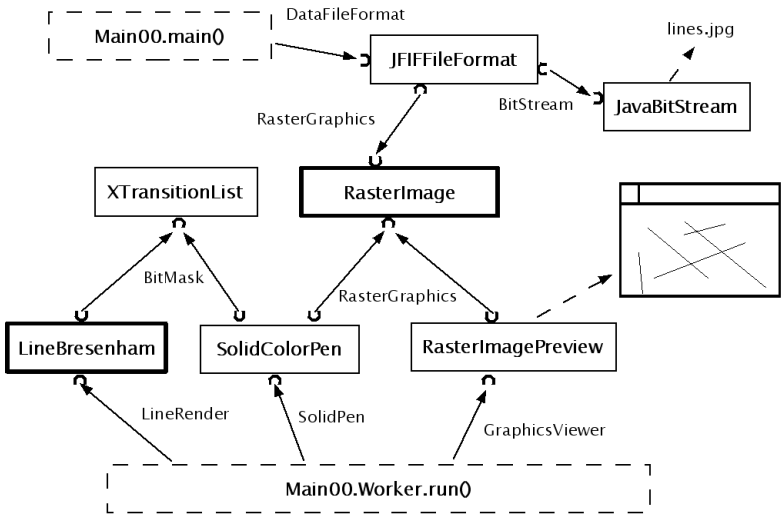

Figure 2. Demonstration of line-drawing algorithm

Module templates

Each module is labeled by one or more templates: they are used to determine overall compatibility between modules. **Template** can be defined as subset of module plugs which forms meaningful interface between module and its neighborhood.

Modules with the same template can be interchanged – this property is very useful both in experimenting and testing phases of algorithm development and in education or demonstration.

Example: "LineRenderToBitMask" is common template for modules which implement line drawing algorithms. Both plugs of "LineBresenham" module (from Fig.1) make the template.

Module parameters

Each module can have one or more public parameters – arbitrary values which can be set from outside of a module. Parameters use to have numerical values (both integral and floating-point), string values, etc.

Relevant interface from core of JaGrLib is named "**Property**" – individual parameters are tagged by string identifiers, values can have virtually any data type available in Java.

GUI environment is able to display parameter values in natural format, giving the user possibility of altering them directly on the screen.

Group of modules

Another important functionality was introduced – set of interconnected modules can be simplified to a **group**. Every group is able to hide its internal implementation to clarify design of the whole composition. Internals of a group can be revealed and edited by an advanced user.

Groups act as regular modules – they have to define plugs, public parameters, they must be registered in the module database, etc.

GUI application appearance

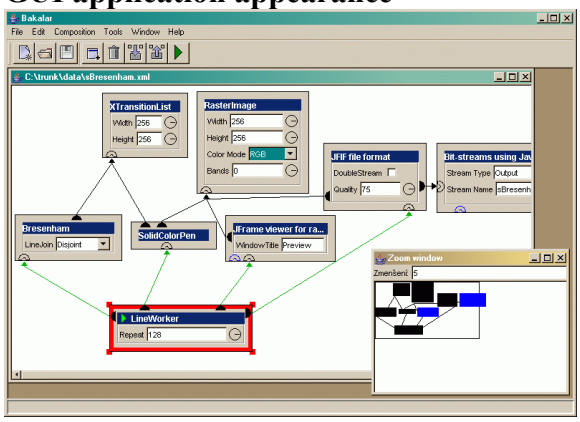

Figure 3. Sample composition - Bresenham

The application has a look of classical MDI frame with number of sub-frames. Many compositions can be opened at the same time – together with some other help frames which can improve arrangement and design transparency.

Examples of GUI configurations are showed in Fig.3 and Fig.4.

Dialog window for selecting of compatible module can be found in Fig.5.

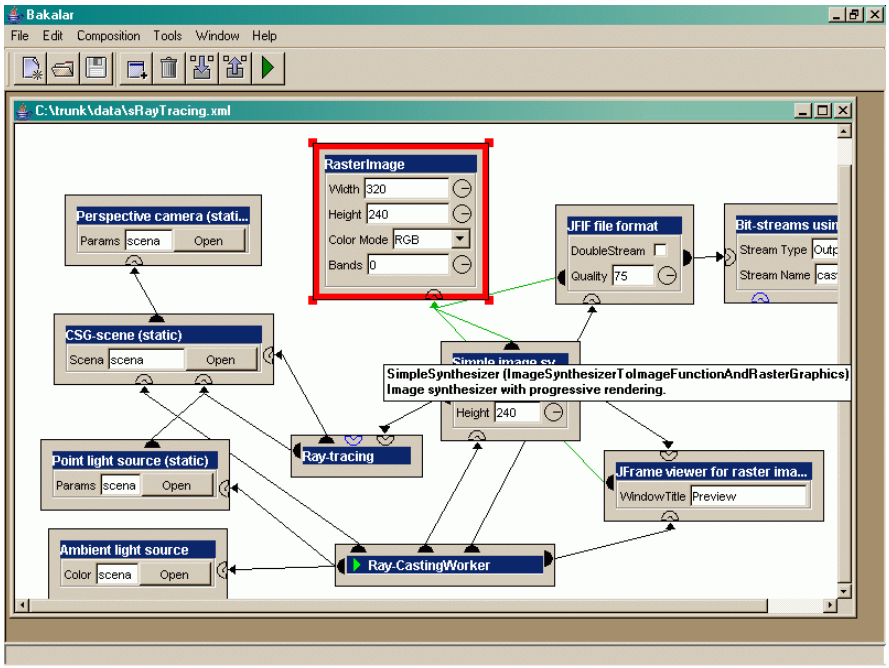

Figure 4. Sample composition - Ray-tracing

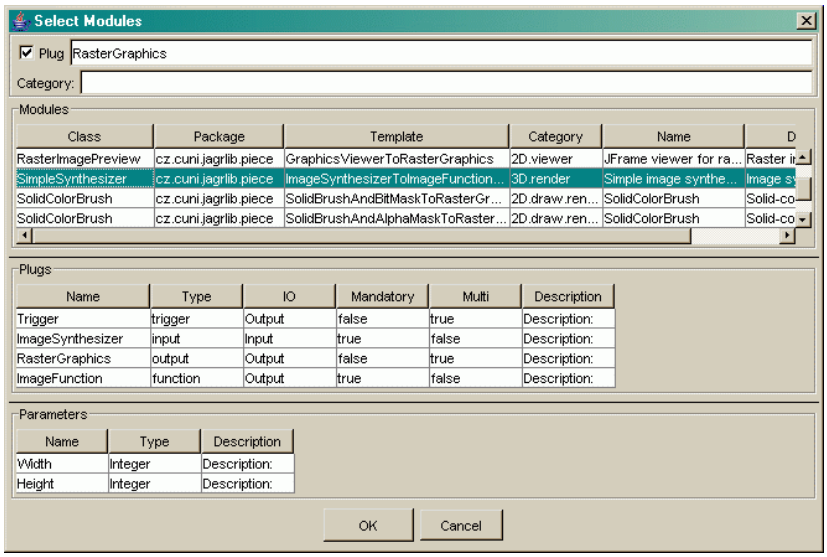

Figure 5. Selection of compatible module

4. IMPORTANT PROTOCOLS

To list all JaGrLib interfaces and modules is beyond scope of this article. We will highlight only a few of them:

- **BitMask**: set of pixels in 2D raster.
- **AlphaMask**: "fuzzy" set of pixels in 2D raster.
- **Pen**, **Brush**: algorithms for "interior filling" of graphical primitives in 2D.
- **RasterGraphics**: storage for single raster image. Pixel formats: gray, color mapped, true-color (RGB), true-color with alpha (RGBA).
- **VectorGraphics**: vector representation of drawing in 2D plane.
- **Brep**: boundary representation for 2D/3D scenes. Can hold polygonal mesh with optional topological information. Any user-configurable attributes can be added to any database entity (vertex, edge, face, solid, division).
- **GeometrySearch**: framework for various geometric-searching algorithms. Defines typical tasks and access methods.
- **Solid**: definition of a solid in 3D (exact representation, used mainly in ray-based rendering methods).
- **LightModel**: set of formulas defining interaction between ray of light and point at solid's surface.
- **ImageSynthesizer**: general conception of raster image synthesis from set of individual point samples.
- **Intersectable**: ability to compute intersection with simple 3D ray.
- **Texture**: any procedure able to modify attributes of single sample point on a solid surface.
- **EntropyCodec**: back-end of most data compression pipelines. Input: finite alphabet, output: stream of bits. Codec is able to switch between different contexts (data models).
- **Order2D**: ordering sequence of 2D lattice.

5. AVAILABILITY

JaGrLib is published under GNU license and can be downloaded from [JaGrLib]. Actual complete source tree is available via SVN system [SVN] at URL:

svn://cgg.ms.mff.cuni.cz/JaGrLib/trunk/

Actual source code is intended to work with Java version 2 (JDK ≥ 1.4.0). JaGrLib does not depend on any third party tool, it can be used alone with Sun's JDK (free download can be found at [JDK]).

6. FUTURE WORK

Many ideas are waiting for implementation, among them we can pinpoint:

- interface to 3D accelerated graphics.
- set of data inspectors and tracers – they will be able to inspect data asynchronously, zoom the graphical representation or trace state changes of a data structure step by step.
- complete implementation of vector graphics pipeline including arbitrary clipping, 2D transformations and font rasterizing.

7. REFERENCES

[JaGrLib] Josef Pelikán, JaGrLib library, WWW pages at http://cgg.ms.mff.cuni.cz/JaGrLib/

[GUI03] Jan Kostlivý, GUI for the JaGrLib library, Bc. thesis, Charles university, 2003

[JDK] JavaSoft WWW pages: JDK documentation and download at http://java.sun.com/

[SVN] Subversion Tigris Source Control System, http://subversion.tigris.org/, http://tortoisesvn.tigris.org/