Klotz: An Agile 3D Visualization Engine

Ricardo Jacas   Alexandre Bergel
Pleiad Lab, Department of Computer Science (DCC),
University of Chile, Santiago, Chile

ricardo.jacas@gmail.com   http://bergel.eu

ABSTRACT

Klotz is an agile 3D visualization engine. Visualizations are produced from an arbitrary model described in terms of objects and interconnections. Any arbitrary model may be visualized. Klotz uses a semi-descriptive scripting language to easily and interactively build visualizations.

Klotz, on its current version, offers four layouts to easily exploit the third dimension when visualizing data.

Klotz is entirely implemented in Pharo. The engine is fully based on the facilities offered by Morphic.

1. INTRODUCTION

Visual displays allow the human brain to study multiple aspects of complex problems in parallel. Visualization, "allows for a higher level of abstract, a closer mapping to the problem domain" [3].

Numerous frameworks have been offered by the Smalltalk community to visualize data. Mondrian[1] [1], one of them, is a flexible and agile visualization engine that uses a two dimensional representation to visualize data.

We are building on the experience we gained with Mondrian by proposing a new visualization engine that adds a third dimension to the graphical representation. Klotz applies the main features of Mondrian, namely the scripting language and the interactive easel, to a new rendering engine.

Contrary to other Smalltalk 3D visualization engines (Lumière[2], Jun[2]), Klotz does not rely on OpenGL or any external libraries. The generation of 3D graphics is solely based on Morph facilities. The benefits are numerous, including ease of installation and multi-platform support.

The paper is structured as follows: Section 2 presents the essential characteristics of Klotz. It progressively presents Klotz' features by giving short and concise illustrative scripts. Section 3 briefly describes the main points of Klotz implementation and gives some benchmarks. Section 5 concludes.

[1]http://www.moosetechnology.org/seaside/pier/tools/mondrian
[2]http://aokilab.kyoto-su.ac.jp/jun/index.html

2. KLOTZ

2.1 Klotz in a nutshell

Klotz is an agile 3D visualization engine. Visualizations are made of cubes and lines. Contrary to other 3D visualization frameworks available on Smalltalk, Klotz maps each graphical element to an object that belongs to an user-defined domain. The visual dimensions of a graphical element is the result of applying metrics on the visualized domain.

Klotz' objective is to offer a flexible and agile tool to visualize any arbitrary domain expressed in terms of objects and relations without any prior preparation. Visualizations are described by means of a scripting language. Consider the illustrative script:

```
1 | subclasses |
2 subclasses := Magnitude subclasses.
3 view nodes: subclasses.
4 view applyLayout: KLSphereLayout new.
5
6 view node: KLEaselCommand using: (KLCube new
     fillColor: Color green).
7 view edges: subclasses from: #yourself to:
     #superclass.
```

Line 1 defines a temporary variable `subclasses`. The variable is initialized in Line 2 with the all the subclasses of the class `Magnitude`. Line 3 adds the objects referenced by `subclasses` into the view. Each of the subclass of `Magnitude` is represented by a cube. Line 4 positions each cube on an invisible sphere.

Line 6 adds a new node, `Magnitude`, the root of the class hierarchy. The node is colored in green. Line 7 adds as many edges there are elements in the variable `subclasses`. For each subclass, an edge is drawn from the subclass to its superclass, `Magnitude`. The result is depicted in Figure 1.

Klotz is intended to be suplementary 3D version of Mondrian, its graphic engine is not based on it tought, but built from the ground. The interface is as similar as it can be to Mondrian's, but its intention is not to bring the same visualizations to 3D figures, but to add new means to analyse code on another perspective, with another dimention to add more information on the object representation itself.

Agility of Klotz is expressed via an easel to interactively "compose" a visualization (Figure 2). The lower part contains the scripts what is entered by the user. The upper part contains the visualization generated by the script interpretation.

2.2 Scripting visualizations with Klotz

The Klotz scripting language is plain Smalltalk code. Each script is based on 4 principles:

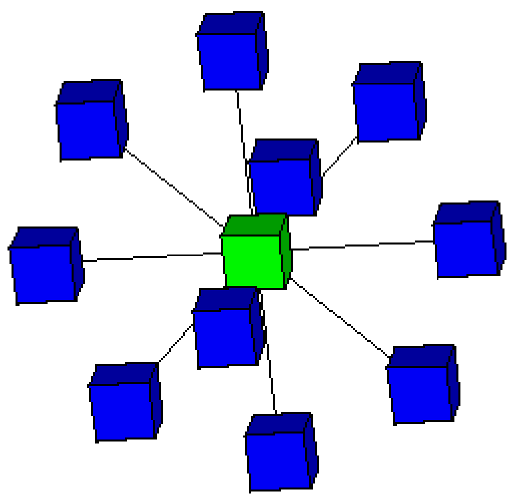


Figure 1: Magnitude subclasses.

- Elements composing the visualization are represented as *nodes*. Each node may have a shape and color that reflect some characteristics of the represented node.
- Relations between elements are represented with *edges*. The color and width of a shape depend on some arbitrary characteristics of connected objects.
- Elements may be ordered using *layouts*. A layout may use edges to direct the ordering.
- Containment is expressed with a *view*, an object that enables the construction of the visualization by offering numerous methods.

The scripting language supports 4 different ways of defining nodes. The message `node:` creates an individual node using some default visual properties (colored in blue, thin black border line). A variant of it is `nodes:` to add multiple nodes in one single instruction. The script

```
view node: Magnitude.
view node: Number.
view node: Time
```

is equivalent to

```
view nodes: { Magnitude. Number . Time}
```

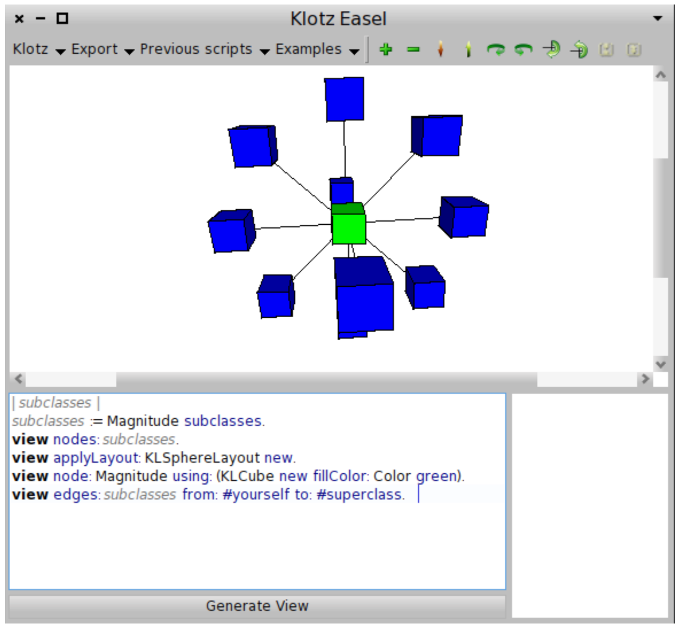
The result is shown in Figure 3.

The visual representation may be particularized according to some characteristics of the provided nodes. The message `node:using:` and `nodes:using:` allow graphical cube to be customized with metrics computed on the represented model. Consider the example, depicted in Figure 4:

```
view nodes: Magnitude subclasses using: (KLCube
    new height: #numberOfInstanceVariables)
```

The message `numberOfInstanceVariables` is sent to each of `Magnitude`'s subclasses. The result of `numberOfInstance-Variables` defines the height of the node.

The methods `edge:from:to:` and `edges:from:to:` use default black and thin line shape offered by the view. Figure 5 shows the result of the script:



```
| subclasses |
subclasses := Magnitude subclasses.
view nodes: subclasses.
view applyLayout: KLSphereLayout new.
view node: Magnitude using: (KLCube new fillColor: Color green).
view edges: subclasses from: #yourself to: #superclass.
```

Figure 2: The Klotz Easel

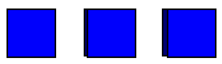


Figure 3: `Magnitude`, `Number` and `Time`.

```
view nodes: Magnitude subclasses.
view edges: (Magnitude subclasses) from: #yourself
     to: MetacelloVersionNumber
```

The message `edges:` *domain* `from:` *fromSelectorOrBlock* `to:` *toSelectorOrBlock* constructs an edge for each element of the *domain*. The source code is the result of evaluating *fromSelectorOrBlock* on the considered node and the target is obtained by evaluating *toSelectorOrBlock*. One-arg blocks and symbols are equally accepted.

2.3 The third dimension

Visualizations in third dimensions convey a "feeling of immersion" that Klotz is intensively exploiting. A number of tools and options are offered by either the visualization or the easel.

Light intensity.

A visualization contains one unique light, a white light located at the same position than the camera. The light intensity on a face is at its maximum when the face is orthogonal to the camera. When a surface of the face is close to be lined up with a light ray, the face is dark.

Emphasizing the perspective.

Perspective is the way a solid object is drawn on a two-dimensional surface so as to give the right impression of their height, width and depth. Our experience shows that it is difficult to precisely compare element positions when closely

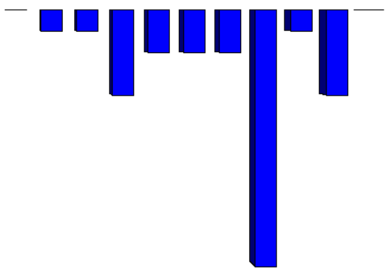


Figure 4: Magnitude subclasses, the heights shows the number of instance variables.

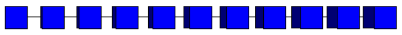


Figure 5: Basic edges

located from each other. Perspective may be emphasized thanks to an increase and decrease perspective commands offered by the easel.

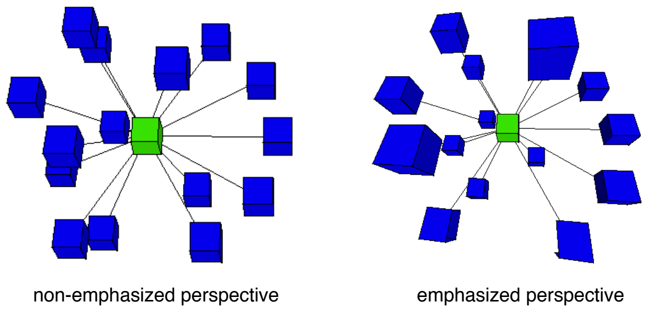


Figure 6: Emphasizing the perspective

Figure 6 illustrates this situation with a slight variation in the perspective.

Controlling the camera.

A user looks at a visualization through the view camera. The easel offers six commands to rotate and move the camera along every axis.

Layout.

Nodes are ordered using a layout. The default layout that is used when no other is specified is the horizontal line layout. A layout is specified using the message `applyLayout:`. Four additional layouts are available: cube layout, sphere layout, block layout, and scatterplot layout.

Cube Layout: This *layout* orders the nodes in a three-dimensional cube.

```
view
   nodes: Magnitude withAllSubclasses
   using: (KLCube new width:
    #numberOfInstanceVariables).
view applyLayout: (KLCubeLayout new).
```

Sphere Layout: nodes are located on the surface of a sphere, centered on the center of the view. The following example places all the subclasses of `Magnitude` on a sphere (Figure 8):

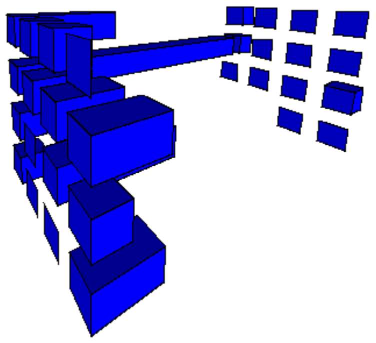


Figure 7: Cube Layout

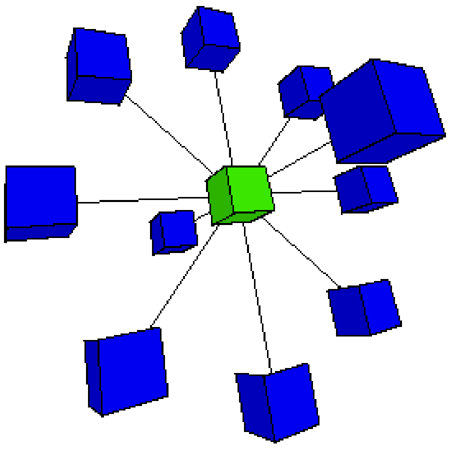


Figure 8: Sphere Layout

```
view nodes: Magnitude subclasses.
view applyLayout: KLSphereLayout new.

view
   node: Magnitude
   using: (KLCube new fillColor: Color green).
view
   edges: Magnitude subclasses
   from: #yourself
   to: #superclass.
```

Block Layout: nodes are hierarchically grouped and organized on a surface. Each group is uniquely colored. The assigned color is randomly chosen if none is specified in the shape.

The following script visualizes the structure of Klotz (Figure 9):

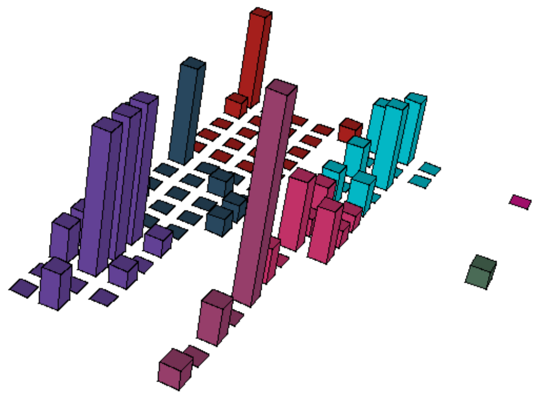


Figure 9: Block Layout

```
| shape packages block |
shape := KLCube new
            height: #numberOfInstanceVariables.
packages :=
   PackageInfo allPackages
      select: [:pak | pak packageName
                  matches: 'Klotz-*'].

block := KLBlockLayout new.
packages do: [ :pak |
   block with:
      {view nodes: pak classes using: shape } ].
view applyLayout: block
```

The script gives a randomly chosen color to each package of Klotz. The color is used to paint the classes of each package. Each node is a colored class. The height represents the number of attributes.

Scatterplot layout: nodes may be located on a three dimensional Cartesian. Each node has a 3d coordinate that is determined from applying three metric on the represented model. The following script plots each class of the Klotz-Kernel package along its number of attributes, number of methods (Figure 10):

```
view nodes:
   ((PackageInfo named:'Klotz-Kernel') classes).

view applyLayout:
(KLScatterCubeLayout new
    "blue line"
    x: [:cls | cls numberOfLinesOfCode / 1000];
    y: #numberOfInstanceVariables; "red line"
    z: #numberOfMethods) "green line"
```

3. IMPLEMENTATION

Klotz is freely available from `http://squeaksource.com/Klotz.html`

The current version of *Klotz*, yet not optimized, provides support up to 1000 *nodes* on screen, as well as, 1000 *edges* between these *nodes*.

As shown in Figure 11, the time taken for the easel to render a scene with the default horizontal line layout is almost proportional with the quantity of nodes and edges.

On Figure 12 illustrates the linear resource taken by the layouts.

The Klotz graphic engine its based on the construction

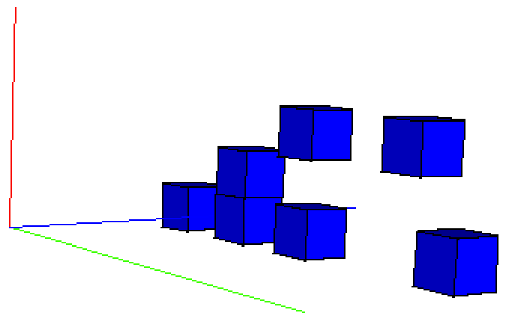


Figure 10: The Klotz Kernel Package

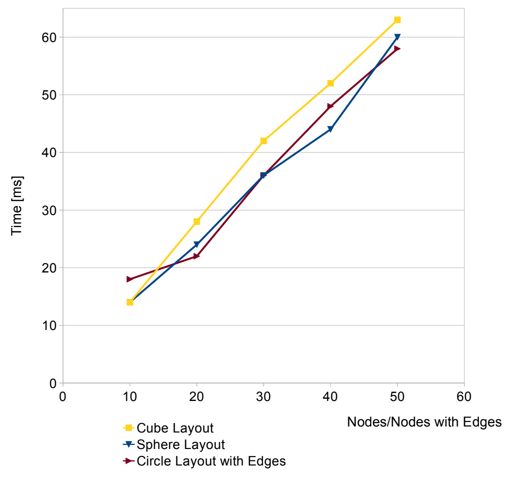


Figure 11: Benchmark (quantity vs time[ms]) for few *nodes*/*edges*

of polygons that are later rendered as regular 2D Morphs. These polygons vertices are calculated using tridimentional vectors with absolute coordinates, and using matrices to produce each transformation (from zooming to the perspective transformation into 2D points). The 3D shapes are also basically optimised, the hidden faces (calculated to some point by need) are not rendered. These calculations, easy as they sound, are often troublesome since calculation for coordinates requires decent, and fast, calculations that sometimes can show perfectly reasonable results in theory (like a point located on the infinity) that must be taken care with proper aproximations, wich must also fix the not accurate aproximations made by the Integer/Float classes.

The Klotz graphic generation solely uses Morph facilities. The most common approach to visualize 3D graphics is to use OpenGL, a reference in the field. We deliberately decided to not use OpenGL for a number of practical reasons:

- OpenGL is distributed as a set of natives libraries, depending on the operating system. Libraries are accessed within Pharo using FFI or Alien, two technologies that interoperate with native libraries. Unfortunately, the recent advances with the Pharo virtual machine significantly reduced the usability of accessing external libraries.

- The visualizations produced with Mondrian rarely go

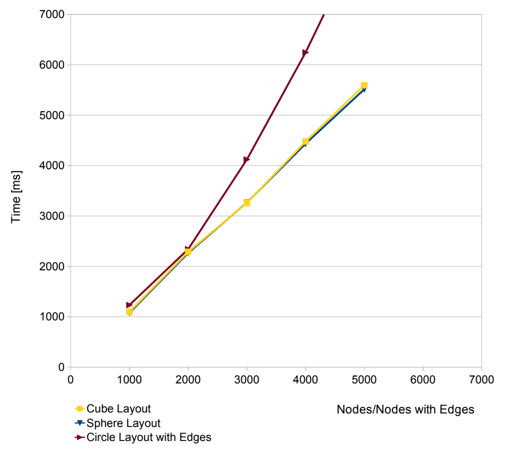


Figure 12: Benchmark (quantity vs time[ms]) for lots of *nodes/edges*

> over 2,000 nodes and 1,000 edges. It is reasonable to expect similar figure as the upper limit for Klotz. OpenGL enables sophisticated rendering, including a high number of rendered polygons and advanced light composition. We do not expect to have such a need in a close future.

Basing Klotz on OpenGL is clearly on our agenda. For this, Alien needs to gain stability, especially with the Pharo JIT virtual machine (Cog).

4. RELATED WORK

4.1 Lumiere

Lumiere [2] is a 3D Framework,that applies a stage metaphor. This metaphor implies that all graphics are produced by cameras taking pictures of 3D shapes lit by the lights of the stage. A stage provides the setting for taking pictures of a composition of visual objects we call micro-worlds. Lumiere's objective is to implement a 3D graphical support to Pharo programmers, at a high level of abstraction. This framework uses OpenGL. As mentioned before, this libraries are entirely dependent of the OS and the right native OpenGL packages within the machine itself, to be used.

On the other hand, Klotz do not represent a competitor on that matter, Klotz is a 3D code visualization engine, its goal is not focus on the actual quality of the graphic interface, but on the expressiveness of the representation in order to achieve a better comprehension of the code. Right now Klotz graphical interface is a small engine that is completely based on the Morph engine. This does not mean to be the final core, eventually the system will need an external graphic engine, probably based on OpenGL, and that can also be Lumiere itself.

4.2 CodeCity

CodeCity [4] is a full fledged city metaphor environment, for code analisys. Its visualization is based on this metaphor, and its metrics are entirely defined (and chosen carefully) to faithfully explain software code. This concrete approach intends to focus not only on the visualization itself, but in the analysis of software evolution. It also support reverse engineering.

CodeCity is programmed in VisualWorks Smalltalk on top of the Moose platform. Just like Lumiere, it uses OpenGL for rendering.

When it comes to Klotz, not been as expressive as Codecity, eventually, and depending on the programer's skills, it can implement most aspect of its model.

5. CONCLUSION

Klotz is an agile three-dimensional visualization engine. Klotz visualizes a graph of objects, without any preparation of the objects. Klotz modeling system allows one to change a graph definition easily, making it simple and fast to adjust a desired visualization. Been based on any object as the items represented on the node it gives great dynamism to the kind of visualization that it can provide. As future work, we plan to:

- Implement drag-and-drop support to manage the nodes easily.
- Add an interface to see the information within the node on mouse focus, and change it dynamically.
- Optimize and improve the graphical libraries, if possible, change to a stable, OpenGL based library.
- Add lots of new Layouts, and with time, other figures to use as nodes.

Acknowledgment.
We thank Patricio Plaza for his effort and participation on an early version of Klotz.

6. REFERENCES

[1] Michael Meyer, Tudor Gîrba, and Mircea Lungu. Mondrian: An agile visualization framework. In *ACM Symposium on Software Visualization (SoftVis'06)*, pages 135–144, New York, NY, USA, 2006. ACM Press.

[2] Fernando Olivero, Michele Lanza, and Romain Robbes. Lumiére: A novel framework for rendering 3d graphics in smalltalk. In *Proceedings of IWST 2009 (1st International Workshop on Smalltalk Technologies)*, pages 20–28. ACM Press, 2009.

[3] Marian Petre. Why looking isn't always seeing: Readership skills and graphical programming. *Communications of the ACM*, 38(6):33–44, June 1995.

[4] Richard Wettel and Michele Lanza. Codecity: 3d visualization of large-scale software. In *ICSE Companion '08: Companion of the 30th ACM/IEEE International Conference on Software Engineering*, pages 921–922. ACM, 2008.