MODELLING AND OPTIMIZATION OF DATA-DRIVEN SCENE GRAPHS

Sébastien Boisgérault
Mines ParisTech, 60 Bd St Michel, Paris, France
Sebastien.Boisgerault@mines-paristech.fr

Eric Vecchié
Mines ParisTech, 60 Bd St Michel, Paris, France
Eric.Vecchie@mines-paristech.fr

Keywords: Modelling, optimization, scene graph, vector graphics, user interfaces, safety critical systems, embedded systems, mobile systems, real-time systems.

Abstract: This article presents data-driven scene graphs, a set of models that address the needs of safety-critical user interfaces design. Data-driven scene graphs merge a description of the user interface behavior as a data-flow program with a description of its graphics content as a hierarchical structure of vector and raster elements. We present a formal description of these models, discuss their semantics and equivalence, and demonstrate that they are suitable for a class of rasterization optimizations based on selective pre-rendering.

1 INTRODUCTION

In transportation systems, the information displays that may influence the vehicle's driving are considered safety-critical. Special efforts shall be made to ensure that these user interfaces are defect-free. Data-driven scene graphs have been introduced in (Esterel Technologies, 2007; Boisgérault et al., 2010) as a class of models that are adapted to this task: they are composed of a functional description and a graphics description using the SCALABLE VECTOR GRAPHICS format (SVG) in our reference implementation. This structure is flexible enough to describe adequately safety-critical user interfaces but has enough constraints to enable validation to be performed at the modelling step. Most defects are therefore eliminated before the code generation stage that actually produces an executable software component.

Efficiency in the graphics rendering is a major issue for such mobile user interfaces that often rely on hardware with limited capabilities. We show in this article that the same type of structuring information that allows validation of data-driven scene graphs also enables optimization to be performed at the model level. Considerable efforts have already been done to design efficient SVG renderers. They either focus on specific hardware architectures (Kim et al., 2010) or optimize some processing steps of software renderers (He et al., 2007). Our approach does not compete but rather complements these efforts as we want first and foremost to address the common situation where the complete graphics platform is already given. In this situation, the only choice left is to optimize the user interface model itself. We also notice that even if some optimizations may be applied to the graphics platform, once a data-driven scene graph model has been transformed into a display list, some model information has been lost. For example, graphics change detection is simple to perform on data-driven scene graphs models (see section 3) but difficult at a later processing stage (Concolato et al., 2008). We present a theoretical framework that enables automatic and safe optimization of data-driven scene graphs.

2 MODEL OF COMPUTATION

The data-driven scene graph Model of Computation consists of a hierarchical description of graphics components driven by a *synchronous program*. Synchronous programming languages are domain-specific languages dedicated to the design of real-time embedded systems. Their semantics are based on the synchronous hypothesis stating that the execution of a

program is discretely divided into atomic reactions. In a reaction cycle, input signals are read/sampled, and internal computation takes place until outputs signals are emitted in answer, and the program state is progressed. In addition to the pure synchronous model, our model consumes its outputs to produce graphics.

2.1 The Synchronous Model

Register Transfer Level (RTL) description is a way of describing the operation of a synchronous digital circuit. Formally, a RTL description is a triplet (S_0, Δ, Θ) that defines relations between the flows of inputs $I_k \in I$, the flows of states $S_k \in S$ and the flows of outputs $O_k \in O$ of the RTL design. The transition function Δ defines the next state of the circuit with respect to the input and the current state of the circuit:

$$S_{k+1} = \Delta(I_k, S_k) \quad \forall k \in \mathbb{N}$$

The output function Θ defines the output flow with respect to the input and the current state of the circuit:

$$O_k = \Theta(I_k, S_k) \quad \forall k \in \mathbb{N}$$

2.2 The Graphics Model

As a graphics extension of the synchronous model, a data-driven scene graph model $H \in \mathfrak{H}$ is formally defined as a quadruplet $H = (S_0, \Delta, \Theta, \Gamma)$ where (S_0, Δ, Θ) is the pure synchronous part of the model and $\Gamma \in \mathcal{G}$ is the graphics function that computes scene graphs G_k with respect to the output of the synchronous model:

$$G_k = \Gamma(O_k) \quad \forall k \in \mathbb{N}$$

The scene graph is then rendered as an image using precise semantics rules. The EDONA/HMI model (Human Machine Interface – Work Package 4, 2008; Boisgérault et al., 2008) is our implementation of data-driven scene graphs. It is an extension of SVG, a family of specifications for describing two-dimensional vector graphics. The data of the graphics model (geometric or style properties of graphics objects) may be exposed as variables while the document structure remains static.

2.2.1 Graphics Tree Grammar

The formal abstract syntax of the scene graph model that captures the SVG model is the following:

$$\begin{array}{rcl} t & ::= & \{n{=}v : t\} \mid v\,?\,t,t \mid t;t \\ & \mid & g[n,...n] \mid \langle t \rangle_e \end{array}$$

with g ranging over graphics primitive names, n ranging over graphics properties names, v ranging over constant values or output signal names and e ranging on environments.

- $\{n{=}v : t\}$ local assignment of graphics properties.
- $v\,?\,t_1,t_2$ if v then t_1 else t_2.
- $t_1;t_2$ compositing of t_2 over t_1.
- $g[n_1,...n_k]$ rasterization of the graphics primitive g using graphics properties $n_1 \ldots n_k$ explicitely.
- $\langle t \rangle_e$ is the raster image that results from the rendering of the graphics tree t in the environment e. This construct defines a unique raster image only when t is *static* – that is when the local assignments in t refer to constant values.

We write $\mathfrak{T}$ the set of trees matching this grammar. We need to distinguish the special cases of group opacity and local transformation matrix. Therefore we will use the notations $\alpha v.t$ and $\mu v.t$ as syntactic sugar respectively for:

$$\{\mathsf{opacity}{=}v : t\} \qquad \text{and} \qquad \{\mathsf{matrix}{=}v : t\}$$

2.2.2 Display List

At each reaction, the data-driven scene graph is reduced into a display list matching the grammar:

$$t \quad ::= \quad g(c,...c) \mid t;t \mid \alpha c.t \mid \mu c.t$$

with g ranging over graphics primitive names and c ranging over constant values and we write $\mathfrak{T}^*$ the set of display lists matching this grammar.

Semantics Rules The reduction of a tree $t \in \mathfrak{T}$ into a display list $t' \in \mathfrak{T}^*$, given an environment E and the outputs O of the synchronous model at a given instant is written:

$$E, O \vdash t \rightharpoonup t'$$

This reduction obeys the following semantics rules:

$$\frac{E[n \leftarrow O[v]], O \vdash t \rightharpoonup t'}{E, O \vdash \{n{=}v : t\} \rightharpoonup t'}$$

$$\frac{O[v] = 1 \qquad E, O \vdash t_1 \rightharpoonup t_1'}{E, O \vdash v\,?\,t_1,t_2 \rightharpoonup t_1'}$$

$$\frac{O[v] = 0 \qquad E, O \vdash t_2 \rightharpoonup t_2'}{E, O \vdash v\,?\,t_1,t_2 \rightharpoonup t_2'}$$

$$\frac{E, O \vdash t_1 \rightharpoonup t_1' \qquad E, O \vdash t_2 \rightharpoonup t_2'}{E, O \vdash t_1;t_2 \rightharpoonup t_1';t_2'}$$

$$\frac{\forall i \in [1..k] \quad E[n_i] = v_i}{E, O \vdash g[n_1,...n_k] \rightharpoonup g(v_1,...v_k)}$$

$$\frac{e, \varnothing \vdash t \rightharpoonup t'}{E, O \vdash \langle t \rangle_e \rightharpoonup t'}$$

Graphics Equivalence Given a RTL model or a data-driven scene graph model:

$$H = (S_0, \Delta, \Theta) \quad \text{or} \quad H = (S_0, \Delta, \Theta, \Gamma)$$

if for all $k \in \mathbb{N}$ and for all input $I_k \in I$ we have:

$$O_k = \Theta(I_k, S_k) \quad \text{and} \quad S_{k+1} = \Delta(I_k, S_k)$$

and if for all environment $E \in \mathfrak{E}$ we have:

$$E, O_k \vdash t_1 \rightharpoonup t_1' \quad \text{and} \quad E, O_k \vdash t_2 \rightharpoonup t_2'$$

then: $t_1' = t_2' \quad \Rightarrow \quad t_1 \overset{H}{\equiv} t_2$

3 SELECTIVE PRE-RENDERING

We propose to optimize the rendering time of data-driven scene graphs by using selective pre-rendering techniques. We want to replace branches of the graphics tree by bitmap images only when this replacement is indeed relevant. Considering our model, it is **not** always **possible** to apply this technique on any branch since some of the graphics parameters are re-computed at each reaction: as an example, consider a shape whose *color* depends on the value of a signal. On the other hand, if the *position* (x,y) of a shape is the sole changing property, then it is possible to save the shape in a bitmap and then combine the bitmap at the expected position.

When possible, it is **not** always **advised** to replace a branch by an equivalent bitmap. As an example consider the figure 1 depicting 2 simple shapes: on the left picture, shapes are quite big with respect to the bounding box and it might be useful to cache such a picture. On the right picture, the 2 shapes are much smaller and far distant: for the same bounding box, the image is quite empty. In this second case, the pre-rendering technique will probably be less efficient than the direct rendering of the image.

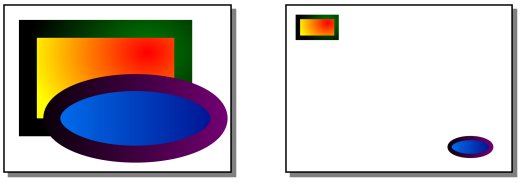

Figure 1: Usefulness of Pre-rendering?

3.1 Minimisation Problem

In our approach, the performances of a model only depends on the graphics operations. The cost of the synchronous part of the system is supposed to be null.

We then define the cost function of the system on the graphics tree. Given a model $H \in \mathfrak{H}$ together with its graphics tree $t \in \mathfrak{T}$ and a cost function $\mathcal{C} : \mathfrak{T} \to \mathbb{R}^+$ we want to compute a tree $t_0 \in \mathfrak{T}$ equivalent to t that minimizes $\mathcal{C}$, i.e. such that:

$$t_0 \overset{H}{\equiv} t \qquad \text{and} \qquad \mathcal{C}(t_0) \leq \mathcal{C}(t') \quad \forall t' \overset{H}{\equiv} t$$

3.2 The Cost Function

Computation time or space is consumed at each instant during the rasterization of the display list. Computation costs have 2 different origins:

1. the **rasterization** cost of graphics primitives
2. the **compositing** cost of bitmaps.

The global cost function $\mathcal{C}(t)$ is obtained by computing the mean of the instantaneous cost $\mathcal{C}_i$ over each reaction of the system:

$$\mathcal{C}(t) = \sum_k \mathcal{C}_i(E_0, O_k, t)$$

The exact cost function is then highly dependent on the input of the considered data-driven scene graph, which prevent us from optimizing the design before it is run. Instead, we use an estimated cost function that is independent on the inputs of the system. This estimated cost can be a theoretical approximation of the graphics primitives costs or even computed on the fly using a sample of representative ouputs of the considered model.

3.3 Minimization Algorithm

The function that minimizes the cost of a tree t_0 using the selective pre-rendering technique is the function $\mathcal{M} : \mathfrak{E} \times \mathfrak{T} \to \mathfrak{T}$. Let $\textbf{cache}?(E,t)$ be the function returning 1 if the tree branch t in the environment E can be cached, 0 otherwise. Let $\textbf{subst}(t, b_1, b_2)$ be the function that returns the tree $t \in \mathfrak{T}$ where the branch $b_1 \in \mathfrak{T}$ has been replaced by the branch $b_2 \in \mathfrak{T}$. Let t_0 be the root of the graphics tree and $O_1, \ldots O_k$ be the output sequence of the model. The function $\mathcal{M}$ is defined as follows. If:

$$\textbf{cache}?(t) \quad \wedge$$
$$\mathcal{C}(\textbf{subst}(t_0, t, \langle t \rangle_E)) < \mathcal{C}(\textbf{subst}(t_0, t, \mathcal{M}'(E,t)))$$
$$\text{then} \quad \mathcal{M}(E,t) = \langle t \rangle_E$$
$$\text{else} \quad \mathcal{M}(E,t) = \mathcal{M}'(E,t)$$

and the auxiliary function $\mathcal{M}'$ is defined as follows:

$$\mathcal{M}'(E, \{n{=}v : t\}) = \{n{=}v : \mathcal{M}(E[n \leftarrow \varnothing[v]], t)\}$$
$$\mathcal{M}'(E, v\,?\,t_1, t_2) = v\,?\,\mathcal{M}(E,t_1), \mathcal{M}(E,t_2)$$
$$\mathcal{M}'(E, t_1; t_2) = \mathcal{M}(E,t_1); \mathcal{M}(E,t_2)$$
$$\mathcal{M}'(E, g[n_1, \ldots n_k]) = g[n_1, \ldots n_k]$$
$$\mathcal{M}'(E, \langle t \rangle_e) = \langle t \rangle_e$$

where $\varnothing[v] = \perp$ if v is not a constant.

4 BENCHMARK

We tested the algorithm on a graphics interface representing the reconfigurable instrument cluster of a car: this includes lights, alert and warning indicators, odometer, speed, RPM, gas and temperature gauges. This model is a typical use case of safety-critical automotive user interface. This graphics interface is shown in figure 2. In this model, every indicator can be either "on" (coloured) or "off" (dark grey), furthermore, the red alert indicators can be scaled big and shown transparent over the speed/RPM gauges (as an emphasis for serious alerts). The odometer is made up of individual led bars (coloured of black) and any needle can rotate around its axis.

Figure 2: The car's instrument cluster model.

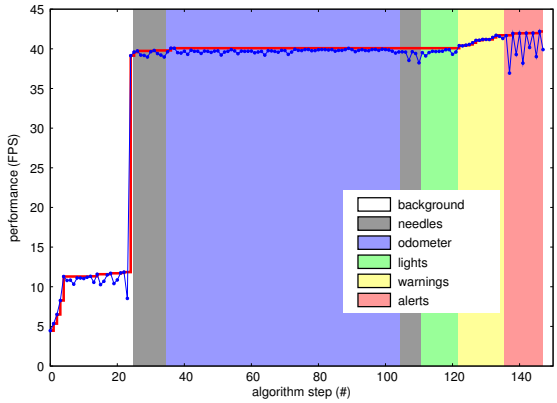

Figure 3: Model optimization process.

The evolution of the pre-rendering algorithm is shown in the graphics of the figure 3. The x-axis represents the steps of the algorithm and the y-axis represents the number of frame per seconds (FPS) that the rasterization were able to achieve.

The cost function were here evaluated experimentally on a 32-bits linux based system equiped with a ATI Mobility Radeon graphics card and using the "image" software rendering backend of the cairo graphics library. Performances were evaluated on a representative set of input values that may influence the rendering performances of the model. The red line represents the evolution of the best performances of the model and the blue dots represent the performances of the current candidate (red line is the "max" of the blue dots).

5 CONCLUSION

We presented a theoretical framework for the modelling of graphics interfaces. We illustrated the usefulness of this framework with a method for an automated optimization of graphics performances. This method optimizes the rendering time of dynamic vector graphics by using a selective pre-rendering technique as evidences show that the systematic pre-rendering is generally not optimal. Groups of vector objects are thus replaced by equivalent bitmaps. This method relies on a cost function of primitive rasterization functions. This includes the cost of group opacity and matrix positioning operations. The global cost function can be evaluated on the fly during the minimization algorithm using a sequence of representative outputs. Experiments show the usefulness of our approach on a concrete safety-critical automotive instrument cluster model. The optimization algorithm benchmarks score a significant improvement of the rendering performances.

REFERENCES

Boisgérault, S., Abdallah, M. O., and Temmos, J.-M. (2008). SVG for automotive user interfaces. In *Proceedings of SVG Open*, Nuremberg, Germany.

Boisgérault, S., Vecchié, E., Meunier, O., and Temmos, J.-M. (2010). EDONA/HMI - modelling of advanced automotive interfaces. In *Proceedings of ERTS*^{2}, Toulouse, France.

Concolato, C., Feuvre, J. L., and Moissinac, J.-C. (2008). Design of an efficient scalable vector graphics player for constrained devices. *IEEE Transactions on Consumer Electronics*, 54(2):895–903.

Esterel Technologies (2007). Scade Display. `http://tiny.cc/scade_display`.

He, G., Bai, B., Pan, Z., and Cheng, X. (2007). Accelerated rendering of vector graphics on mobile devices. volume 4551, pages 298–305.

Human Machine Interface – Work Package 4 (2008). EDONA HMI Format. Report, EDONA.

Kim, S., Oh, Y., Park, K., and Ro, W. (2010). Hardware implementation of a tessellation accelerator for the OpenVG standard. *IEICE Electronics Express*, 7(6):440–446.