

Informatics 134

Software User Interfaces
Winter 2022

Mark S. Baldwin

baldwinm@ics.uci.edu

1/18/2022

Agenda

1. This Week
2. Basic Structured Graphics
3. References

This Week

This Week

- Lecture on Thursday
- Launch A3 and T2 on Thursday
- Keep working on A2/T1

Basic Structured Graphics

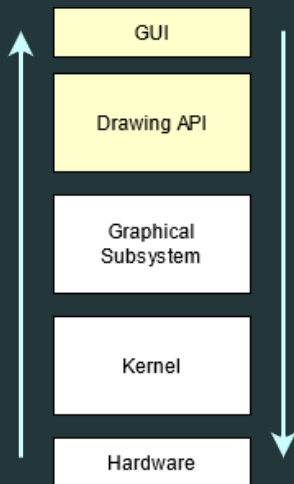
Basic Structured Graphics

Review the Requirements of a Graphical Program

Manage what gets rendered to a screen

Manage how it gets rendered

Manage when it gets rendered



Requirements of a Graphical Program

As programmers of graphical user interfaces, we primarily concern ourselves with *what* is rendered, rather than *how* or *when*.

Any thoughts on why?

Requirements of a Graphical Program

The *how* and *when* are largely repeatable tasks that do not change across different user interfaces.

The *how* and *when* requirements, therefore, can be abstracted into reusable mechanisms that can support the *what* that programmers create.

This type of system is called "Structured Graphics"

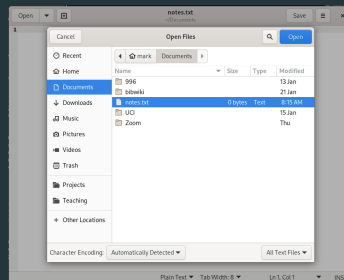
Basic Structured Graphics

Structured Graphics System

Encapsulate a primitive (rectangles, lines, images, icons, etc.)

Expose reusable code for rendering how and when.

Enable programmer to create the what (e.g., a button)



Advantages of Structured Graphics

- Less code, more reusable

- Encapsulation of common mechanisms enables automation of required actions like redraw and refresh

- Hierarchical model supports custom encapsulation as well

Some Trade-offs

Supporting reuse increases memory consumption

Redraw and refresh can take more time

Combined, can effect 'snappiness' of UI

Though modern computing power negates most of these concerns

Redraw and Refresh Operations

Operation depends on underlying algorithm (the how and when)

One approach is to redraw every object every time a change occurs to any graphical object. Trade-offs?

Draws all objects in the hierarchy from back to front (from a display perspective), top down hierarchically

Redraw and Refresh Operations

Operation depends on underlying algorithm (the how and when)

Another approach is to only redraw the area of the display that has changed.

Trade-offs?

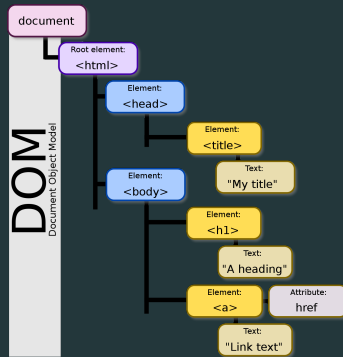
Capture all objects that intersect the area to be redrawn, redraw from back to front.

Basic Structured Graphics

Structured Graphics System

The DOM used in web browsers is an example of structured graphics.

Maintains a hierarchical list, or “retained object model” of all graphical objects. Update the screen by editing objects in the list.



[Eriksson, 2022]

Basic Structured Graphics

The Hierarchical List

Graphical Primitives

text, icons, and shapes

Aggregates

collections of graphical objects

“div” or ‘Groups’ in SVG

Parent/child relationship

The Hierarchical List

Hierarchies are built through aggregate object types and inheritance.

One Example: CSS.

```
1 body {  
2     color: green;  
3 }  
4 .my-class-1 a {  
5     color: inherit;  
6 }  
7 .my-class-2 a {  
8     color: initial;  
9 }  
10 .my-class-3 a {  
11     color: unset;  
12 }
```

```
1 <ul>  
2     <li>Default <a href="#">link</a> color</li>  
3     <li class="my-class-1">Inherit the <a href="#">link</a> color</li>  
4     <li class="my-class-2">Reset the <a href="#">link</a> color</li>  
5     <li class="my-class-3">Unset the <a href="#">link</a> color</li>  
6 </ul>
```

The Hierarchical List

Issues and Design Considerations

- Complexity increases with features

- Which point in the hierarchy has responsibility for a given property

- Which hierarchy is responsible for themes? events propagation?

- Should objects change in appearance based on type or should each type be a new object?

A2 or T1: Any Questions Yet?

References

References i



Eriksson, B. (2022).
Document object model.



Mozilla (2021).
Cascade and inheritance.