Informatics 134

Software User Interfaces
Spring 2023

Mark S. Baldwin
baldwinm@ics.uci.edu
4/12/2023

Agenda

Upcoming

- Lecture today
- Launch A1 and T2
- T1 Due tonight
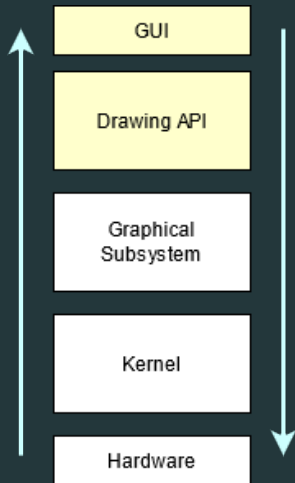
Basic Structured Graphics

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

GUI

Drawing API

Graphical Subsystem

Kernel

Hardware

Requirements of a Graphical Program

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

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.

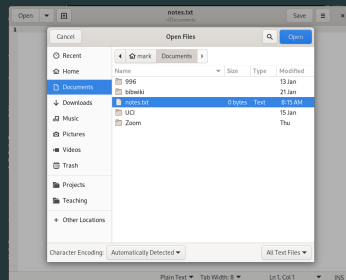This type of system is called "Structured Graphics"

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)

Basic Structured Graphics

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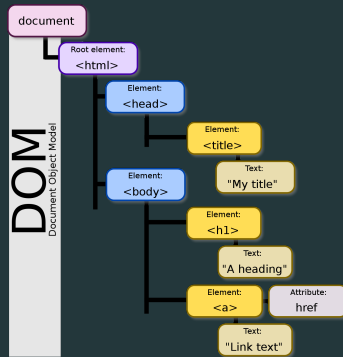>
> Capture all objects that intersect the area to be redrawn, redraw from back to front.

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]

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.

```css
body {
    color: green;
}
.my-class-1 a {
    color: inherit;
}
.my-class-2 a {
    color: initial;
}
.my-class-3 a {
    color: unset;
}
```

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

[Mozilla, 2021]

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?
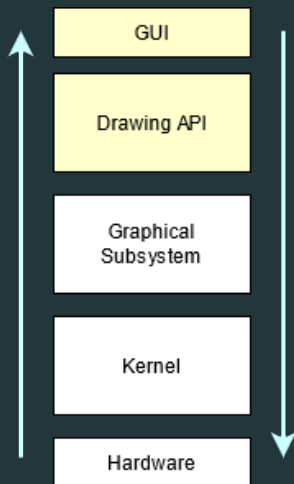
Application of Structured Graphics

Application of Structured Graphics

Build your own graphical toolkit?

Ground up using 'low-level' graphics primitives

Work in the browser (avoid intricacies of OS)

Build on event propagation model of the DOM (rather create our own!)

GUI
Drawing API
Graphical Subsystem
Kernel
Hardware

Application of Structured Graphics

Two Options:

Drawing

- Objects retain specification after draw
- Objects can be moved, changed, and scaled
- Better suited for user interfaces, high resolution
- In browser: Scalable Vector Graphics (SVG)

Painting

- Objects become pixels after draw
- Editable through change
- Better suited for games/graphics, low or fixed resolution
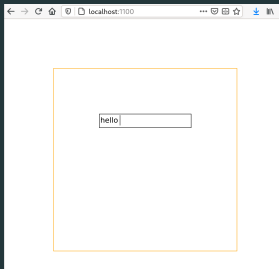- In browser: Canvas

We will be using SVG

SVG.js (https://svgjs.dev/)

No dependencies–fast

Choose your own editor or IDE

Hierarchy in SVG

Let's start with a look at hierarchy in raw SVG using SVG.js

```javascript
import {SVG} from './svg.min.js';

SVG.on(document, 'DOMContentLoaded', function(){
    var draw = SVG().addTo('body').size('1000px','1000px');
    var window = draw.group();
    window.rect(400,400).stroke("orange").fill("white");

    var group = draw.group();
    var rect = group.rect(200, 30).fill("white").stroke("black");
    var text = group.text("hello").move(2,4);
    var caret = group.line(45, 2.5, 45, 25).
        stroke({ width: 1, color: "black" });

    group.move(100,100);

    window.add(group);
    window.move(100,100);
});
```

SVG and the DOM

Which language?

- Javascript or Typescript?
- Typescript is built on strong types and object oriented principles
- Javascript does not require transpiling

SVG and the DOM

Object Oriented Model

- Already popular with GUI toolkits
- Conceptually similar to primitives and aggregates
- Easier (IMO) to reason about how a hierarchy should be structured.

Object Oriented Hierarchy in Typescript

Abstraction

Class == graphical object

Instances

Inheritance

```typescript
interface WidgetState{
    ...
}

class Window implements WidgetState{
    ...
}
```
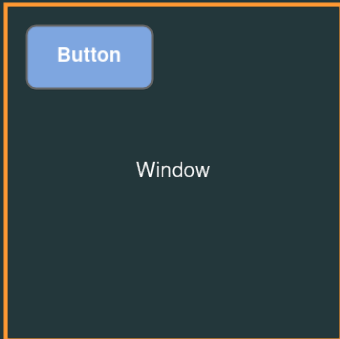
Object Oriented Hierarchy in Typescript

Abstraction

Class == graphical object

Instances

Inheritance

```typescript
abstract class Component implements IAccessibility {

}

class Widget extends Component{
    constructor(){
        super(100, 50);
        ...
    }
}
```

```
let w = new Window(500,500);
let btn = new Button(w);
btn.backColor = "blue";
```

Object Oriented Model

Already, you can see how decisions need to be made.

- Design and optimize classes and interfaces to avoid duplicate code (costly space/performance)
- Encapsulate when possible (the Button class should not need to worry about backcolor change)
- Sensible names and parameters

Individual Assignment 1: Roll Your Own Button

Individual Assignments

You will create the following widgets (and more!):

- Button (use starter code, customize)
- Check Box
- Radio Button
- Text Box
- Scroll Bar
- Progress Bar
- Custom (your choice)

Individual Assignment 1: Roll Your Own Button

You will be responsible for all of the following:

- Apply a custom theme across all of your widgets
- Implement features
- Create a state chart for each widget
- Create a small GUI program that makes use of all of your widgets

Individual Assignment 1: Roll Your Own Button

Getting started:

- Full assignment is upon the course website
- Start looking at the SVG.js documentation
- Start working on your button
- We will be covering more over the next few weeks

Let's Dive In!

References

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

📄 Mozilla (2021).
Cascade and inheritance.