Follow @derekknox

Chapters
+

- Environment
- Mindset
- Subset
- Keywords
- Expressions
- Operators
- Statements
- Functions
- Errors

Chapter 6 Deconstructing Designs

- Process
- Deconstruct
- Reconstruct Structure
- Reconstruct Style
- Reconstruct Behavior

Light
-

Chapter 2 Structure, Style, & Behavior

V
isual designs created directly with code or computer software most often only use 2D space. Do not let this be a limitation as a designer. Designs leveraging 3D space can be extremely valuable too. Especially when designing novel user interfaces, games, and other creative interactive products. Ultimately, 3D design excels when conveying spatial and depth relationships is valuable.

Though a 2D and 3D design may look drastically different in comparison, a computer utilizes the same three concepts to render animated and interactive visuals for each. These concepts are:

1. Structure
2. Style
3. Behavior

The virtual world in addition to the real world leverage these same concepts in fact.

A building for example (virtual or physical) is composed of a particular arrangement of a set of building blocks. This is structure. These building blocks are adorned with materials and other objects that have aesthetic or functional value. This is style. The building blocks of the structure in combination with the adorned materials and objects may respond to interaction, the environment, and time. This is behavior.

Combined, structure, style, and behavior enable an interactive design to exist, evoke meaning for, and engage one or more viewers. Exist. Evoke. Engage.

In the following sections we will visit each concept from both a 2D and a 3D perspective. We will do so in the context of the web platform for 2D design and the Unity® platform for 3D design. This approach will help solidify your general mental model for coding, help crystalize gaps in knowledge, and expand your design thinking into 3D space. Right on.

It is worth noting that when I refer to 3D design, I refer to an authoring environment that natively supports 3D positioning, rotation, and scaling of objects. A 3D rendering ultimately becomes a 2D image, so the authoring environment is the distinctive aspect.

#

Structure

Structure, as previously mentioned, is a particular arrangement of a set of building blocks. In simpler terms, structure is organization. Structural designs are the organization of structural building blocks.

These building blocks are contextual. In the context of 2D with the web platform, these building blocks are HTML elements (HyperText Markup Language). In the context of 3D with Unity, these building blocks are GameObjects.

The building blocks are different depending on the context, but the underlying relationships between them are identical. Reusable knowledge is the win here. These relationships are referred to as *parent-child* relationships, a *tree*. In both a 2D and a 3D structure, a tree of parent-child relationships exists. The same parent-child relationships—the same tree structure—can have many layout variations in both 2D and 3D. Structure is not layout, it is simply the tree of parent-child relationships.

The *body* for an HTML document and the *scene* for Unity are each the root visual parent for their respective platforms. The simplest tree structure, in both 2D and 3D, consists of only this root parent. This is a one-node tree.


One-Node Tree

Pretty boring eh? Here is what that looks like for HTML:

```
<body></body>
```

Yup, still boring. The second simplest structure is a two-node tree. This is when the root node has a single child node.


Two-Node Tree

Still boring, but less so. The third simplest structure has two variants:

1. Two child nodes
2. A single child that is also a parent due to itself containing one child

Now it is getting interesting. These are both examples of three-node trees.


Three-Node Trees

The structural variations building off these examples, as you can imagine, are infinite.

It may be surprising, but the only difference between 2D and 3D structural trees are the building block types associated with the nodes. The variations of the parent-child relationships are identical for 2D and 3D. What results is still a tree structure. This is the takeaway.

The computer program—the web browser for our 2D examples and the Unity Engine for our 3D examples—knows how to decode a tree structure. Each program then outputs a 2D or 3D rendering respectively. It is important to understand that the tree structure simply defines the relationships between the building blocks. HTML elements for 2D with the web platform and GameObjects for 3D with the Unity platform.

As our custom defined structure is respected and properly reflected by each platform, we may begin to stylize each building block. This stylization process ensures particular aesthetic or functional effects are present for the viewer. As visual designers, this is usually the most fun and rewarding part. Style time.

#

Style

Style, as previously mentioned, is where the building blocks of a structure are adorned with materials and objects that have aesthetic or functional value. An adornment allows a specific building block or a group of building blocks to look or function identically. An adornment may also ensure a building block is truly unique in its look or function. Each platform may have default styles for its building blocks but ultimately the creative power is in your hands as the coder and designer.

These building block adornments are contextual. In the context of 2D with the web platform, these adornments are applied via CSS styles (Cascading Style Sheets). In the context of 3D and Unity, these adornments are Materials.

The adornments are different depending on the context, but their underlying purpose is identical. Their purpose is to provide aesthetic or functional value to one or more building blocks. There is a wildly large gamut of creative output that can result in both 2D and 3D by combining various adornments. Your creativity and experience are the only limiting factors.

In either the 2D or 3D context, it is not important to know all the possible adornments. Professional coders simply know the purpose of style and its distinction from structure and behavior. A professional's level of experience simply makes him or her quicker in their application of it. Professional coders—just like beginners—still need to reference resources. For this reason, we will use only a very small subset in the examples below. You can reference resources to expand your understanding of what is possible.

In the 2D CSS context, this is done by adorning an HTML element with a specific *class*. The class has a unique name that you as the coder create. Within the class definition, you list one or more property-value pairs. Each pair simply defines a style property and a value for it. Really straightforward. We'll go into detail later, but let's see a CSS class definition in action.

```
.dark-mode {
    background-color: #272822;
    color: #EEEEEE;
}
```

In fact, if you scroll to the top and toggle the "Light -" button, you'll see this code get applied to the page. That is, if you haven't already. I additionally use a transition so the colors change smoothly over time. Again, you can reference resources to learn what is possible so don't get hung up on the details.

Ultimately, if valid properties and values are set, the browser does your bidding and renders your design. If any are invalid, the browser ignores your invalid code.

To review the code snippets above, we first define a structure. Then, we adorned specific building blocks of the structure with style. Simple as that. We do the exact same thing in 3D with Unity.

In the 3D Unity context, we adorn a GameObject with a material. A material in Unity, like a CSS class, has a set of properties that can be assigned certain values. In Unity you can set these property values via code or through the UI of the Unity editor. It's worth noting that 2D CSS and 3D materials don't always have the same properties, but they often do under a similar name.

2D CSS code example:

```
background-color: red;
```

3D Material code example:

```
material.color = Color.red;
```

As you'll notice in the 2D and 3D examples above, what differs between them is only the property names and values. The concept of adornment for applying style remains constant. This is the takeaway.

The computer program—the web browser for our 2D examples and the Unity Engine for our 3D examples—knows how to decode valid properties and values. Each program can then output a stylized 2D or 3D rendering respectively.

As style is respected and reliably applied by each platform, we begin to truly differentiate the visual representation of potentially identical structures. With just the one node tree example we can already apply over 16 million different `background-color`s for that many differentiated visuals. This is a super powerful concept to grasp so etch this in your brain.

> As style is respected and reliably applied by each platform, we begin to truly differentiate the visual representation of potentially identical structures.

Traditional visual design ends after the application of structure and style. The examples showcased thus far, when rendered, are static by default. Though there is nothing inherently wrong with this fact, we can do more as designers. We can add behavior to a design and quite literally bring it to life. This is an exciting aspect and a powerful ability to attain. Welcome to coding behavior.

#

Behavior

Behavior, as previously mentioned, is where the structural building blocks in combination with their adornments *react*. They react to interaction, the environment, and ultimately time. Behavior enables one or more building blocks and one or more adornments to change in time. These changes manifest in an infinite amount of ways.

The possible behaviors, and their respective ease of application, is contextual. In the context of 2D with the web platform, behavior is applied via JavaScript code. In the context of 3D and Unity, behavior is applied via C# code. Each coding language, in conjunction with each platform's authoring tools, provide ways to help you apply certain behavior. For example, Unity with C# provides simple ways (compared to the web and JavaScript) to apply behavior in 3D space.

Behaviors may be different depending on the context, but their fundamental purpose remains constant. Behavior brings life to a design. It does so in the form of interactivity, motion, and logic. For example, a design with behavior may manifest as a creative authoring environment, an entertaining animation, or a mission critical tool. The manifestations are endless and limited only by your imagination.

Let's explore some specific applications of behavior by building off the previous structure and style examples.

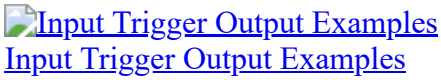
We know behavior may be applied to structure (a specific building block or group of building blocks). Additionally, we know behavior may be applied to style (adornments). The sheer presence of behavior however, due to its dynamic nature, results in an additional target we have yet to talk about. Time itself. Time in the form of delays and schedules are most common.

Now is a good time to explicitly identify example triggers that result in behavioral reactions. A trigger is *input* and the reaction produces *output*. Without input triggers a design remains static and lacks dynamism. It lacks life. There are three types of input triggers:

1. User interaction (tap, click, hover, gesture, voice, etc.)
2. Environment (layout resizing, operating system, device sensors, etc.)
3. Time (delays, schedules, etc.)

As you can imagine, the output possibilities are endless as a result of any of the above input triggers. This is where you get to be super creative.

New building blocks or groups of building blocks may be added to the structure dynamically. They can also be removed dynamically. Adornments may be added or removed dynamically too. Behavioral changes to these structures or adornments are just as easily modifiable.


Input Trigger Output Examples
Input Trigger Output Examples

The takeaway is that there are three input trigger types that provide an opportunity for output behavior. The specific inputs and outputs are up to you, the creator.

Where traditional visual design is static, behavior makes a design dynamic and interactive. This power is addictive, useful, and extremely empowering. In the context of the web platform and Unity, instead of using solely what the JavaScript and C# code provides, you can *extend* each language's capabilities with your *own* code. You can invent entirely new and better ways to do almost anything. Super cool.

Next we will look at the pieces that makeup the ability to *code behavior*. We will do so in the familiar context of the Elements, Principles, and Constructs of Visual Design. Rock on!

Chapter 1 Breaking Barriers

Chapter 3
Programming & Visual Design