DAY 1

Vijay Shula, "Flash Professional CC: The Future of Animation"

Flash CC provides 'platform independent creativity': you can copy content from HTML <canvas>, WebGL, or any 3rd party platform. Developers can create their own plugin for Flash Pro. To add a plugin you need to create/implement a few APIs, package it as an extension, and click to share. As an end user you just need to install the extension. Flash CC can publish to a number of platforms: Flash player, AIR, <canvas>, WebGL, and and third-party library you care to implement.

New tools in Flash CC:

- variable width stroke tool let's you animate line width
- a motion editor is available in the timeline, and you can apply motion tweens to 85 different properties.
- a number of default rushes and more coming in the near future, and simple custom brush functionality already available

Flash CC let's you create accessible interactives at speed.

Bermon Painter, Jason Pamental, Jeff Francis, Alex Blom, "Why You Can Start Using Web Components Now"

A snapshot of panelist's position:

Bermon Painter: It's good for enterprise and helps keeps things DRY.

Jason Pamental: Helps you create components to keep your design modular. There's an interesting potential to have things that totally encapsulate in a single component what you're thinking in terms of the user experience. They'll help break design into legos blocks, which is an interesting idea to explore.

Jeff Francis: Web components enables rapid prototyping.

One main concern with web components is that there's currently nothing making sure they be made and used in an accessible way (although you can and should). Bruce Lawson has done a lot to bring this to the fore.

For those trying to learn web components, the spec is readable (webcomponents.org), and there's yeoman generator for a polymer setup to get you started. (And there's nothing wrong with starting with polymer; for now they're a "necessary evil".) To get started, try finding a repo not using web components and refractor to use web components.

In general the panelist don't recommend using WC today since there's not enough support yet, but do recommend working in a way that is componentized in order to train the behaviour, e.g. source everything into pattern library. Although, If you only needed to worry about Chrome when developing, you "can totally rock WC". If you must use it in production, build ontop of a smaller library polyfill.

Bermon Painter, "Fostering Collaboration: Pattern Libraries / Rapid Prototyping"

Writing clear, consitent code that you can pick a year later is a big challenge.

A component/module:

- can be nested
- can be combined with other modules
- is clearfixed (we're still dealing with floats a lot)
- is flexible or responsive
- separates structure from look and feel
- must be accessible

A drawback of a pattern library is that it takes a lot of time up front to develop, test, document, and educate developers and bosses. Communicate the benefit of pattern libraries to these stakeholders: there are huge efficiency gains to using things that have already been considered, and have passed unit tests. Pattern lets you focus on whether building the right thing instead of getting distracted by implementation details. Add components to your pattern library as you go/need them.

Your pattern library should show the HTML code and a fully functional example. Explore tools like Slim, an HTML preprocessor, if you're interested in creating a pattern library.

Examples of pattern libraries:

- cardinal solutions pattern library on Github.
- Lonely Planet is closest to doing something that has a single source of truth, a true "pattern API"

Andrew Smyk, "Digital Kids on Branding, Privacy and Technology Bias"

People 2-3 years apart in age are having completely different technology experiences. The tech given to kids is preselected by parent they are given the cheaper touch technology as a hand-me-down and by-and-large the love it. The Apple and Google brands are going to become the equivalent of tissues and sneakers to these kids. They don't have brand loyalty though, they're just there for the content. Kids will prefer e-readers over print media for the multimedia, unless a caretaker gets involved (kids prefer to engage with the parent and whatever the caretake is using). Generally the concept of digital is disappearing – it's not an ebook, it's simply a book. Kids move back forth between digital and analogue easily.

DAY 2

Nick Van Weerdenburg, "Lean UX Lessons Learned from One Dozen Projects"

A primer on lean UX: It's iterative development process. UXD is the practise of design and development of the user experience. Siloing design and development can be harmful, the optimal learning experience occurs at the intersection. A central issue today is that we underestimate the complexity and effort required to build a user experience. We now see any project as 1/3 UX, 1/3 core app logic development, (I missed 1/3 of something here).

Follow the 80/20 rule: work on 20% of the project (assuming it's likely to correspond to 80% of the users time in app), get feedback, and then focus on the next 20%. Beware: design docs created over time eventually become development standards almost as a necessity since no one can otherwise remember why they did things. Accordingly, software documentation begets a waterfall process over time as the documents replace conversation. There are psychological effects to having docs; they do affect how you work. Having to have those conversations again takes time – its like redoing the process. The solution is to shorten the link between work and validation so the conversations do not need to be preserved in documentation. Do much less UX up front – do it as you go instead – so as not to create document that has apocryphal authority (is treated as a spec). Always treat upfront UX as a hypothesis and deliver continuously.

The process:

Client research (Avoid: tends to spiral out of control) -> Market definition (may not be needed, discuss and narrow your focus) -> User research (drive down into fundamental interactions) -> Test (testing removes opinion, otherwise everone on the team has a slightly different take on should be (as is) built)

1. Market defining (depends on where the project is in its life cycle)
2. Interactive prototyping (RIP static prototypes)
3. Low-resolution and a few high-resolution mockups
4. Style guide
5. Develop working software (small and tight) for testing

Steps 1 - 4 can take as little a day, and no more than a week with an average team size of 5 ppl. In total, a project lasts 3 - 4 months.

Tom Emrich, "Marketing Opportunities with Wearable Tech: Today and Tomorrow"

It's the 'dawn of a new wave of computing'.

We're moving from the information age, (internet is collective knowledge base), to augmenting our memory. We have created a reflection of the physical world in the digital world. All our memory is placed in the digital world, but it isn't a true reflection. That's because we actively need to created the digital space; we need to feed it. We're in the age of awareness, of "waking things up". We're waking things up so they will passively supply data to the digital world. This means more datasets and less work for us. Our wearable tech will be gathering information about the real world 24/7, allowing the digital world to more accurately reflect the physical world. We will benefit by then allowing computers to make decisions on our behalf. The fridge will collude with the pantry and our wearable tracking device to match our calorie intake with a meal plan, and then use Amazon's Dash to deliver groceries to the front door.

Currently there's a separation gap between us and out tech: we put our smartphones down when we're done with them. We take our VR goggles off. As a result, there's a disconnect between ourself and the self reflected in the digital world. This causes us to be distracted: when we're trying to share information in the digital space we're less present in the real world. We're trying to do two things at once and the separation between us and the tech is the cause.

Wearing our tech closes the separation gap. Wearable tech allow us to enjoy a digital moment and physical moment at the same time, not taking away from either, e.g. using a wearable camera that passive take photos instead pulling out our smartphone, launching instagram....

Devices will also augment our natural senses by allowing new connections to be made to the world, e.g. experience the war zone in Syria from our bed. The most immediate opportunity for wearable tech is gathering data. Gathering step count is good way to start gathering biometrics, but the goal is now to get more advanced data. What we really want to know is how a user is feeling and reacting. We can measure arousal state to infer emotion for example. That's useful data we currently don't have as a marketer. We could use that data to better profile users and tailor campaigns. We want to create an experience that feels like kismet: suggesting what you need when you need it. Brands are already experimenting with wearable tech, e.g. Disney's Magic Band.

"Software always proves the value of hardware."

This talk was fucking awesome.

Rami Sayar, "What's new in ES6 for Web Devs"

ES6 is targeted for ratification in June 2016 and is now available under a harmony tag in Node.

A few highlights:

- Block scoping: currently scoping is at the function level. 'Let' and 'Const' will give use block-level scoping. Let is bound to lexical environment instead of variable environment. Let = no more variable leak!
- Destructuring: var [var, foo] = ['foo', 'bar']; Not limited to arrays. Can do pattern matching: you can pass an object to function and retrieving value for key, e.g. function ({a; 1}) {do stuff}
- Classes: class skinnedMesh extends THREE.mesh { }
- Modules: implicit asynchronous loading, but won't execute until all modules are loaded. Can import all, or selected functions. You can automatically, lint, compile (and more) modules on import.
- Iterators and generators: an iterator will be an object that has a next method. A generator is factory for iterators.

Jessica Rosenkrantz, "Growing Objects"

Jessica combines architecture, biology, computer science and math. Her projects typically start with a natural inspiration, e.g. writing an algorithm for the veins of a leaf the lead to physical applications. For example, Hyphae is a 3D printed lamp. Each lamp is unique, since "variation is free" with digital fabrication. With programming, each 3D printed object can truly be one of a kind.

For her Floraform projects, all the forms we see are an event in space-time, not merely a configuration in space. Take inspiration from process based design.

Chip Kidd, "! or ?"

Chip creates a book for teaching graphic design for kids (incidentally, I've read it as an adult). It breaks graphic design into essential parts: form, scale, foreground/background, focus, simplicity/complexity, positive/negative space, typography. A typography 101 pro-tip: take a word and make it look like it is what says what it is. Irony is when you flip that around.

When to be literal (!) in design:

- safety at stake
- to educate (debatably)

When to be mysterious (?), ask your self

- how much do you think your audience wants to know?
- how much do they already know? (If it's a lot, you can afford to be mysterious.)
- is there already a lot of buy in

Question to self: Is there ever a time to be mysterious in scientific illustration? Presenting a visual 'mysteriously' could be engaging and have improved learning outcomes due to the learner actively trying to solve the mystery.

DAY 3

Louis Lazaris, "Front-end Tools: Sifting Through the Madness"

A snap-shot of different front-end tools:

HTML & CSS

- HTML inspector: provides output in the console and you can write your own rules
- HTML code sniffer: specify the accessibility test you want
- DOM flags: create boomarks in your HTML to go between them quickly

- UCSS: cleans up unused CSS styles
- Bear CSS: upload HTML and it will spit out a CSS selector skeleton
- CSSanimations: provides a clean API for building CSS animations with JS
- ai2HTML: converts illustrator files into HTML and CSS
- CSS2SCSS: tool that converts CSS to SASS which might be useful if you're just getting into sass.
- DOMenlightenment book

JS

- ReFiddle: Fiddle for RegExp
- JSONselector: helps write selectors for deeply nested JSON data
- Background check: changes colour on the fly depending on the value of element underlying, ensure minimum contrast
- HTML.js: helps define HTML selectors
- JavaScript design patterns: a free, online book

Editors

- Floobits: code with another developer while not in the same room (pair-program)
- Wake time: tracks time for projects open in your text editor as you switch between them
- Octotre: chrome extension for navigating chrome repositories

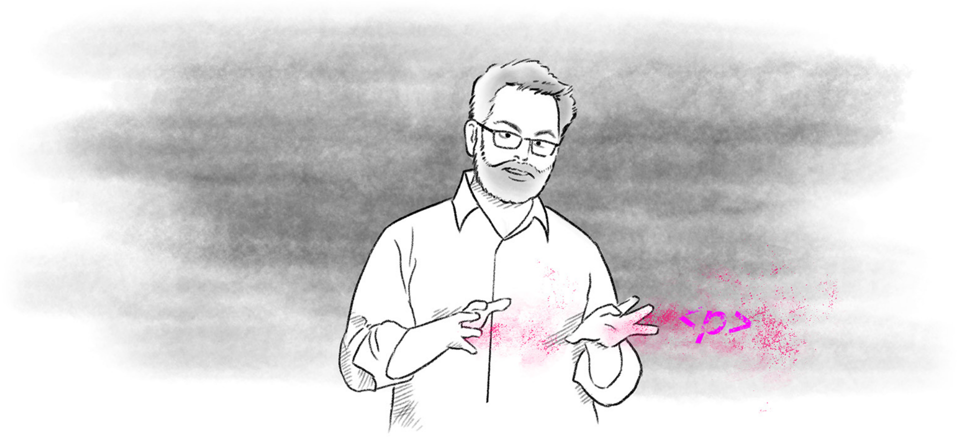
Grant Skinner, "Building the Dragon Age Interactive Story Summary"

CreateJS encompases a number of creative libraries and was used to build an interactive story summary for Dragon Age. They used Flash for animations and exported to HTML5 canvas (they developed the plugin). In the past HTML5 animations would result in double or triple the budget, but now it is financially viable to build rich web content without a lot of struggle. However, the graphics can be more limited in canvas compared to video.

Process

- Received layered Ai files from Bioware,
- Pulled content from Illustrator into Flash
- Used Flash to animate the content
- Set <canvas> export to dump to CreateJS

The resulting animations were kinda 'flat' so they built a virtual camera system to add camera depth. For responsive design, they used two cameras with two centres of focus. Complex vectors, masks, gradients etc. can get really slow. To improve performance they cached bitmaps. They developed tools to track performance and got browsers to scale to 1.5x for better performance. For different languages, stretched the animation timescale or introduced hold/pause points. The final result Rruns on desktop, console, tablet, mobile, has performance scaling, is resolution independent, has dynamic aspect ratio, which helped validate the use of CreateJS for the project.



Jason Pamental, "The Life of <p>"

"Type is the clothes words wear"

Jason started our journey for a great typographic system by starting at the <3, the paragraph. Egyptians underlined a character to indicate that it began a new thought. Greeks insert 'pilcrow' character to indicate breaks in thought or paragraph. With the advent of printing, couldn't keep up with inserting special characters, so you just left a space (indents). Enter the typewriter: it was easier to hit return twice then return and indent, resulting in double line break. Our modern paragraph style. This helped with scalability.

Some pro-tips:

- Orphans: How do we avoid the dreaded orphan line? We can use javascript to fix this, for now, e.g. widowmaker.js.
- Hyphenation: Prefer a ragged right with hyphenation to no hyphens or justified (rivers), see the JS library Hyphenator for help with hyphenation.
- Dropcap: Use pseudo-selectors to create these beauties.
- Bold-first line: p:first-line pseudoselector
- Link: The holy grail: lightweight lines that break around the descencender. Some have achieved this using a white text shadow.
- Ligatures: Jason has a mixin for that.
- Loading external fonts: DO NOT USE font-squirrel to convert a desktop font!
- Copy: Don't ever design with Lorem Ipsum, use unfinalized copy.