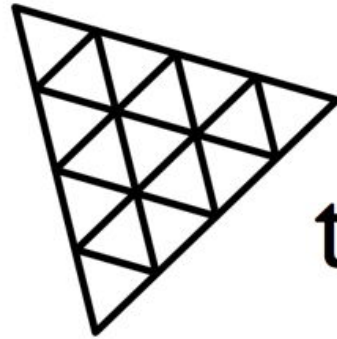
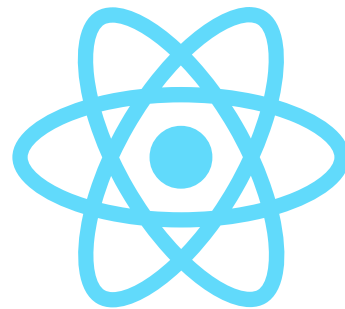


THREE REASONS



three.js

REASON



OUR MISSION

- Learn some ReasonML
- Build a cool 3D thingy
- Decide if it's Un-ReasonML?

```
1.
2. 'use strict';
3.
4. var Three = require("./three");
5. var Three$1 = require("three");
6. var CrateGif = require("./textures/crate.gif");
7.
8. function init(element) {
9.   var rect = element.getBoundingClientRect();
10.  var scene = new Three$1.Scene();
11.  var renderer = new Three$1.WebGLRenderer();
12.  var camera = new Three$1.PerspectiveCamera(70.0, rect.width / rect.height, 1.0, 1000.0);
13.  camera.position.set(0, 150, 400);
14.  var texture = new Three$1.TextureLoader().load(CrateGif);
15.  var material = new Three$1.MeshBasicMaterial({
16.    map: texture
17.  });
18.  var geo = new Three$1.BoxGeometry(125, 125, 125);
19.  var cube = new Three$1.Mesh(geo, material);
20.  scene.add(cube);
21.  renderer.setSize(rect.width, rect.height);
22.  element.appendChild(renderer.domElement);
23.  var playing = [/* true */1];
```

OCaml My Caml

- Rich type system
- FP + OOP
- Cross platform compilation

Bucklescript & ReasonML

- Compiles to JS
- JS-like syntax
- NPM workflow

Why should I care?

- Real type safety for your JavaScript
- Excellent foreign function interface
- [React ready](#)

Please Bear With Us...




```
type state = {
  initialized: bool,
  playing: bool
};
type action =
  | Toggle;
let component = ReasonReact.reducerComponent("App");
let make = (_children) => {
  ...component,
  initialState: () => { initialized: false, playing: false },
  reducer: (action, state) =>
    switch action {
    | Toggle => ReasonReact.Update({ ...state, playing: !state.playing })
    },
  render: (self) => {
    let buttonText = self.state.playing ? "Pause" : "Play";
    <div className="App">
      <button onClick={_ => self.send(Toggle)}> {ReasonReact.stringToElement(buttonText)} </button>
    </div>;
  }
};
```

Working with the DOM

- Using React refs
- Handling nullable data
- Styling in reason-react

Actions: before

```
type action =  
  | Toggle;
```

Actions: after

```
type action =  
  | Ready(option(Dom.element))  
  | Toggle;
```

**No Null
References!**

Reducer: before

```
reducer: (action, state) =>  
  switch action {  
    | Toggle => React.Update({ ...state, playing: !state.playing })  
  },
```

Reducer: after

```
reducer: (action, state) =>
  switch action {
  | Toggle => ReasonReact.Update({ ...state, playing: !state.playing })
  | Ready(canvas) => ReasonReact.SideEffects(_ => {
    switch (canvas) {
    | (Some(c)) => Js.log(c)
    | _ => ()
    }
  })
  },
```

Render: before

```
render: (self) => {  
  let buttonText = self.state.playing ? "Pause" : "Play";  
  <div className="App">  
    <button onClick={_ => self.send(Toggle)}> {ReasonReact.stringToElement(buttonText)} </button>  
  </div>;  
}
```


Render: after

```
render: (self) => {  
  let buttonText = self.state.playing ? "Pause" : "Play";  
  <div className="App">  
    <button onClick={_ => self.send(Toggle)}> (ReasonReact.stringToElement(buttonText)) </button>  
    <div ref={c => self.send(Ready(Js.Nullable.toOption(c)))} />  
  </div>;  
}
```

```

type state = {
  initialized: bool,
  playing: bool
};

type action =
  | Ready(option(Dom.element))
  | Toggle;

let component = ReasonReact.reducerComponent("App");
let make = (_children) => {
  ...component,
  initialState: () => { initialized: false, playing: false },
  reducer: (action, state) =>
    switch action {
    | Toggle => ReasonReact.Update({ ...state, playing: !state.playing })
    | Ready(canvas) => ReasonReact.SideEffects(_ => {
      switch (canvas) {
      | (Some(c)) => Js.log(c)
      | _ => ()
      }
    })
    },
  render: (self) => {
    let buttonText = self.state.playing ? "Pause" : "Play";
    <div className="App">
      <button onClick={_ => self.send(Toggle)}> {ReasonReact.stringToElement(buttonText)} </button>
      <div ref={c => self.send(Ready(Js.Nullable.toOption(c)))} />
    </div>;
  }
};

```

CSS in JS?
Sorry, not yet.

How about inline styles?
Ok

Optional labeled arguments

```
let canvasStyle = ReactDOMRe.Style.make(~height="100vh", ());
```

```
type controller = { playPause: unit => unit };
let init = _ => {
  let playing = ref(true);
  {
    playPause: () => {
      let nextState = !(playing^);
      Js.log(nextState ? "Playing" : "Paused");
      playing := nextState;
    }
  }
};
```

State: before

```
type state = {  
  initialized: bool,  
  playing: bool  
};
```

State: after

```
open Game;
type state = {
  initialized: bool,
  playing: bool,
  controller: option(controller)
};
```


Actions: before

```
type action =  
  | Ready(option(Dom.element))  
  | Toggle;
```

Actions: after

```
type action =  
  | Start(controller)  
  | Ready(option(Dom.element))  
  | Toggle;
```

Reducer: before

```
reducer: (action, state) =>
  switch action {
    | Toggle => ReasonReact.Update({ ...state, playing: !state.playing })
    | Ready(canvas) => ReasonReact.SideEffects(_ => {
      switch (canvas) {
        | (Some(c)) => Js.log(c)
        | _ => ()
      }
    })
  },
```

```

open Game;
type state = {
  initialized: bool,
  playing: bool,
  controller: option(controller)
};
type action =
  | Start(controller)
  | Ready(option(Dom.element))
  | Toggle;
let component = ReasonReact.reducerComponent("App");
let canvasStyle = ReactDOMRe.Style.make(~height="100vh", ());
let make = (_children) => {
  ...component,
  initialState: () => { initialized: false, playing: false, controller: None },
  reducer: (action, state) =>
    switch action {
    | Start(c) => ReasonReact.Update({ initialized: true, playing: true, controller: Some(c) })
    | Toggle => ReasonReact.UpdateWithSideEffects({ ...state, playing: !state.playing }, self => {
      switch (self.state.controller) {

```

```
type scene;
type renderer;
type domElement;
type childObject;
type camera = childObject;
type geometry;
type material;
type mesh;
type vector;
type loader;
type texture;
type materialSpec = { . "map": texture };
```

Defined As:

```
[@bs.new] [ @bs.module "three" ] external boxGeo: (int, int, int) => geometry = "BoxGeometry";
```

Used like:

```
let geo = boxGeo(125, 125, 125);
```

Compiles to:

```
var geo = new Three$1.BoxGeometry(125, 125, 125);
```


Defined as:

```
[@bs.new] [@bs.module "three"] external textureLoader: unit => loader = "TextureLoader";  
[@bs.send.pipe : loader] external load : string => texture = "load";
```

Used like:

```
let texture = textureLoader().load("path/to/texture.gif");
```

Compiles to:

```
var texture = new Three$1.TextureLoader().load("path/to/texture.gif");
```

Defined as:

```
[@bs.get] external getDomElement : renderer => domElement = "domElement";
```

Used like:

```
let gameCanvas = renderer |> getElement;
```

Or maybe like this?

```
let gameCanvas = getDomElement(renderer);
```

Compiles to:

```
var gameCanvas = renderer.domElement;
```

```
/* Our types! */
type scene;
type renderer;
type domElement;
type childObject;
type camera = childObject;
type geometry;
type material;
type mesh;
type vector;
type loader;
type texture;
type materialSpec = { . "map": texture };
/* Constructors */
[@bs.new] [@bs.module "three"] external newScene: unit => scene = "Scene";
[@bs.new] [@bs.module "three"] external newRenderer: unit => renderer = "WebGLRenderer";
[@bs.new] [@bs.module "three"] external newCamera: (float, float, float, float) => camera = "PerspectiveCamera";
[@bs.new] [@bs.module "three"] external boxGeo: (int, int, int) => geometry = "BoxGeometry";
[@bs.new] [@bs.module "three"] external basicMeshMaterial: materialSpec => material = "MeshBasicMaterial";
```



```
1.
2. open Three;
3.
4. [Obs.module] external crate : string = "./textures/crate.gif";
5.
6. type controller = {
7.   playPause: unit => unit
8. };
9.
10. let init = element => {
11.   let unwrapped = ReactDOMRe.domElementToObj(element);
12.   let rect = unwrapped##getBoundingClientRect();
13.
14.   let scene = newScene();
15.   let renderer = newRenderer();
16.   let camera = newCamera(70.0, rect##width /. rect##height, 1.0, 1000.0);
17.   camera |> setPosition(0, 150, 400);
18.
19.   let texture = textureLoader() |> load(crate);
20.   let material = basicMeshMaterial({ "map": texture });
21.   let geo = boxGeo(125, 125, 125);
22.   let cube = newMesh(geo, material);
23.
24.   scene |> add(cube);
25.
```

THREE REASONS WHY

- First rate type system for JavaScript
- Easy integration with Node projects
- The language is cool AF

THREE REASONS WHY NOT

- Immaturity
- High velocity
- Added complexity

THANKS!