Functional Programming vs Efficient Computer Graphics

GPU Day, 21-22 June 2018

Attila Szabo
Georg Haaser
Harald Steinlechner

Outline

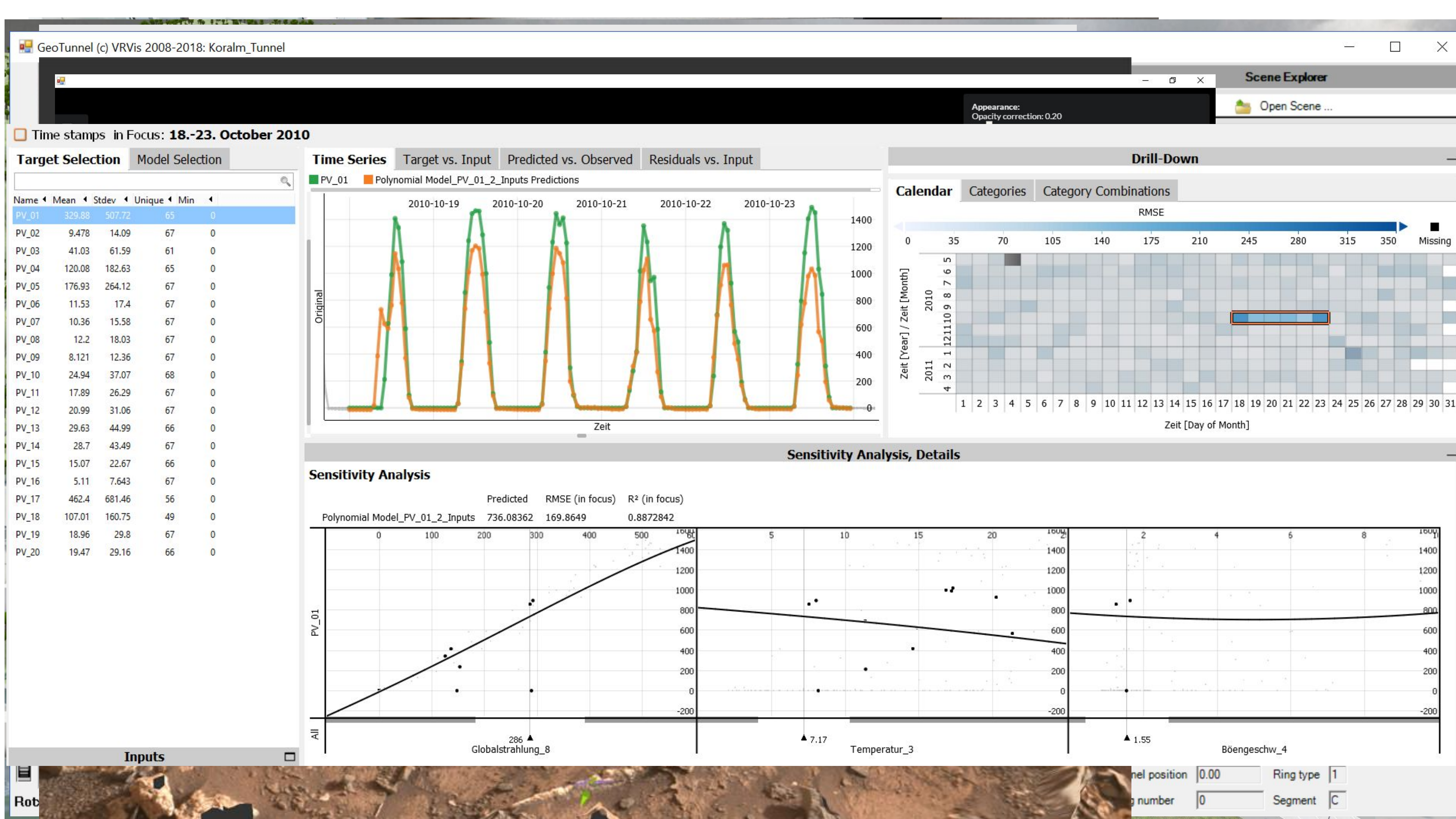
- Introduction to our Institute/Team/Projets

- Rendering Engine Experience
  - Functional Programming (**FP**) in High-Performance Visual Computing
- 4 parts
  - FP for **photogrammetry**
  - FP for **efficient rendering**
  - FP for **shader programming**
  - FP for **in real projects**

} by using
domain specific languages (DSLs)

Takeaways

- Functional Programming in High-Performance Visual Computing
- Domain Specific Languages help
- Real World functional programming
  - Experience from moving to purely functional


GeoTunnel (c) VRVis 2008-2018: Koralm_Tunnel
Scene Explorer
Open Scene ...
Appearance:
Opacity correction: 0.20
Time stamps in Focus: 18.-23. October 2010
Target Selection
Model Selection
Name Mean Stdev Unique Min
PV_01 329.88 507.72 65 0
PV_02 9.478 14.09 67 0
PV_03 41.03 61.59 61 0
PV_04 120.08 182.63 65 0
PV_05 176.93 264.12 67 0
PV_06 11.53 17.4 67 0
PV_07 10.36 15.58 67 0
PV_08 12.2 18.03 67 0
PV_09 8.121 12.36 67 0
PV_10 24.94 37.07 68 0
PV_11 17.89 26.29 67 0
PV_12 20.99 31.06 67 0
PV_13 29.63 44.99 66 0
PV_14 28.7 43.49 67 0
PV_15 15.07 22.67 66 0
PV_16 5.11 7.643 67 0
PV_17 462.4 681.46 56 0
PV_18 107.01 160.75 49 0
PV_19 18.96 29.8 67 0
PV_20 19.47 29.16 66 0
Inputs
Time Series
Target vs. Input
Predicted vs. Observed
Residuals vs. Input
PV_01
Polynomial Model_PV_01_2_Inputs Predictions
2010-10-19
2010-10-20
2010-10-21
2010-10-22
2010-10-23
Original
Zeit
Drill-Down
Calendar
Categories
Category Combinations
RMSE
Missing
Zeit [Year] / Zeit [Month]
2010
2011
Zeit [Day of Month]
Sensitivity Analysis, Details
Sensitivity Analysis
Predicted
RMSE (in focus)
R² (in focus)
Polynomial Model_PV_01_2_Inputs
736.08362
169.8649
0.8872842
PV_01
All
286
Globalstrahlung_8
7.17
Temperatur_3
1.55
Böengeschw_4
Ring type
Segment

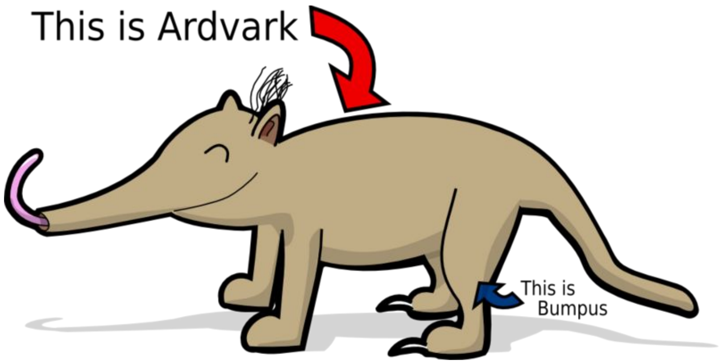
What is Aardvark?

shared development platform for projects

research platform for visual computing

...

open source project on github



https://commons.wikimedia.org/wiki/File:Ardvark_The_Aardvark_Original.png

2005 — **First Aardvark in C#**

2008 — **Composability/Flexibility**

- *Semantic Scene Graph [Tobler 11]*
- Easy to use
- Expressiveness

2010 — **Performance**

Lazy Incremental Computation For Efficient Scene Graph Rendering [Wörister et al. 2013]

2014 — **Domain Specific Languages**

- Composable Shaders [Haaser et al. 2014]
- Attribute Grammars

2015 — **Performance + Dynamism**

- Incremental Rendering VM [Haaser et al .2015]

2017

- **Vulkan**
- **Purely Functional App Frontend**

Concepts

- High-Level abstraction via
  - Domain Specific Languages
  - Functional Programming

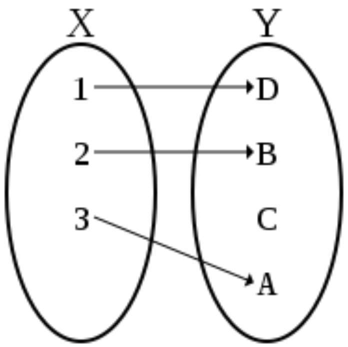
```
Sg.sphere 5 color size
 |> Sg.shader {
      do! DefaultSurfaces.trafo
      do! DefaultSurfaces.vertexColor
      do! DefaultSurfaces.simpleLighting
 }
 |> Sg.trafo trafo
```

- High-Performance via
  - Compilers
  - Incremental Evaluation
  - Low-Level trickery

```
let inline rexAndModRMSIB (wide : bool) (left : byte)
              (rex : byref<byte>) (modRM : byref<byte>) =
    let r = if left >= 8uy then 1uy else 0uy
    let w = if wide then 1uy else 0uy
    rex <- 0x40uy ||| (w <<< 3) ||| (r <<< 2)

    let left = left &&& 0x07uy
    modRM <- 0x40uy ||| (left <<< 3) ||| 0x04uy
```

Part I

Functional Programming vs. Efficient Computer Graphics




```
MOV ECX, 0xb90
MOV RAX, 0x7a..
CALL RAX
```

https://bar.wikipedia.org/wiki/Datei:Injection.svg

Renderer:

```
If attributes.Dir
UpdateGpuData()

Foreach spawned geometry do
   addAndOptimize
```

Asnyc Loading

Save/Load State

GameModule:

```
If mouse.IsDown
then
   updateGui()
   startRocket(
```

Out of core

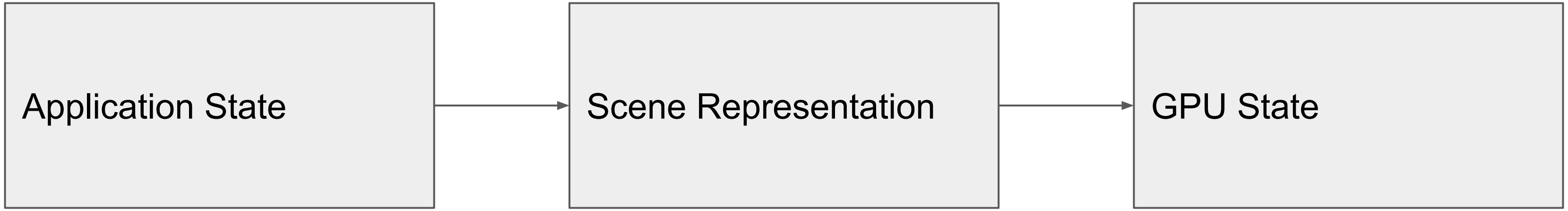
Undo Redo

Character Rendering

```
If character.hasMoved() then
  asyncUpdateLevelOfDetail()
```

Rendering Engine Challenges

- Performance
- Synchronization between modules



- Side effects -> complexy

Why use Functional Programming?

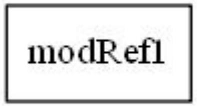
- Pros
  - Easier reasoning/debugging
  - No side effects
  - concurrency
  - parallel programming
  - Features such as
    - Persistency
    - Undo Redo
- Cons
  - **Performance** (?)

Purely Functional	?	Mutable imperative
Parallelism Reasoning Persistence		Performance Memory usage In-place updates Algorithms

A Functional approach to Mutation

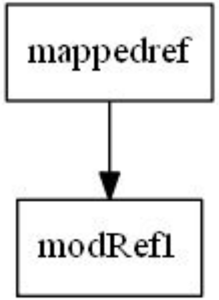
- Creates modifiable input cell

```
let modRef1 = Mod.init 10
```



- Create single edge dependency

```
let mappedref = Mod.map (fun s -> s + 1) modRef1
```



- Mod.force evaluates a dependency graph

```
Mod.force mappedref ⇒ 11
```

Dependency Graph Operations

- Dependency Graph = Directed acyclic graph
- Feed changes into system

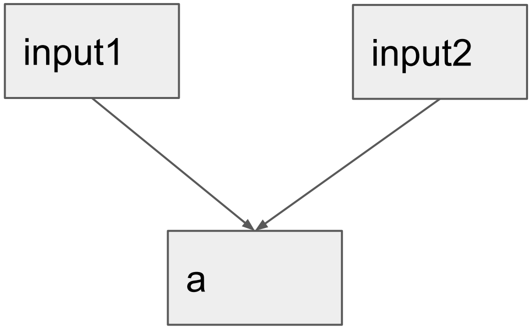
```
transact (fun () ->
    Mod.change modRef1 0
)
```

- Extract current state from the system

```
Mod.force mappedref ⇒ 1
```

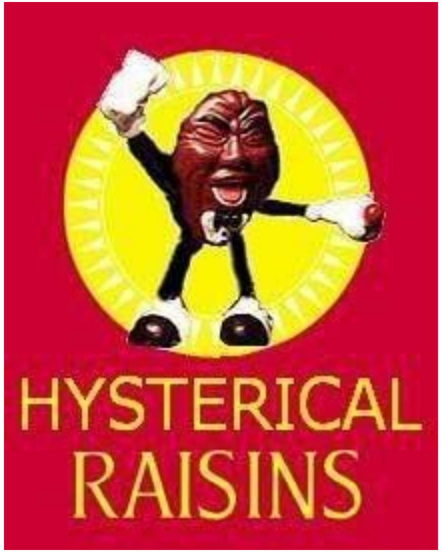
Basic operations can be hidden beneath DSL

```
let input1 = Mod.init 10
let input2 = Mod.init 20
.....
let a =
   adaptive {
      let! currentInput1 = input1
      let! currentInput2 = input2
      return m + c + d
   }
```



Approach: Monads for incremental computing [Carlsson 2002]

- Theoretically well-founded
  - Adaptive Functional Programming [Acar 2002,2005,...Hammer et al 2014]
- ModRef = changeable input values
- IMod = dependent value
- Extends to sets
  - cset = changeable set
  - aset = dependent set
  - Lists, maps….

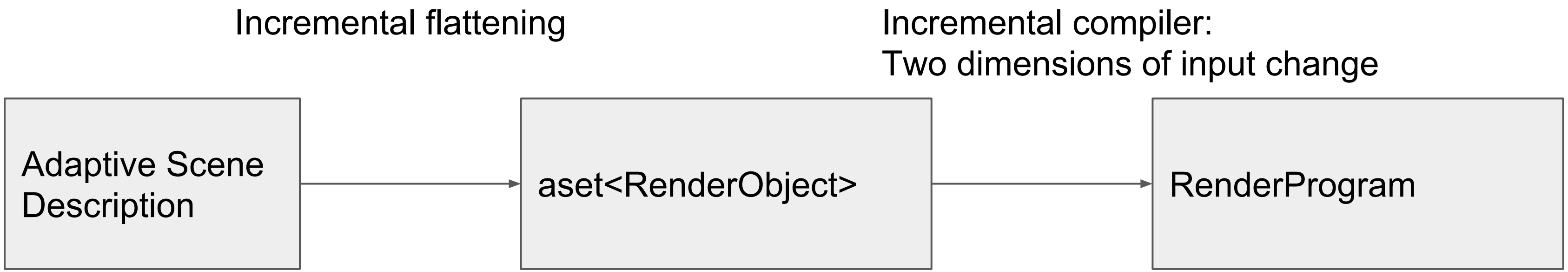


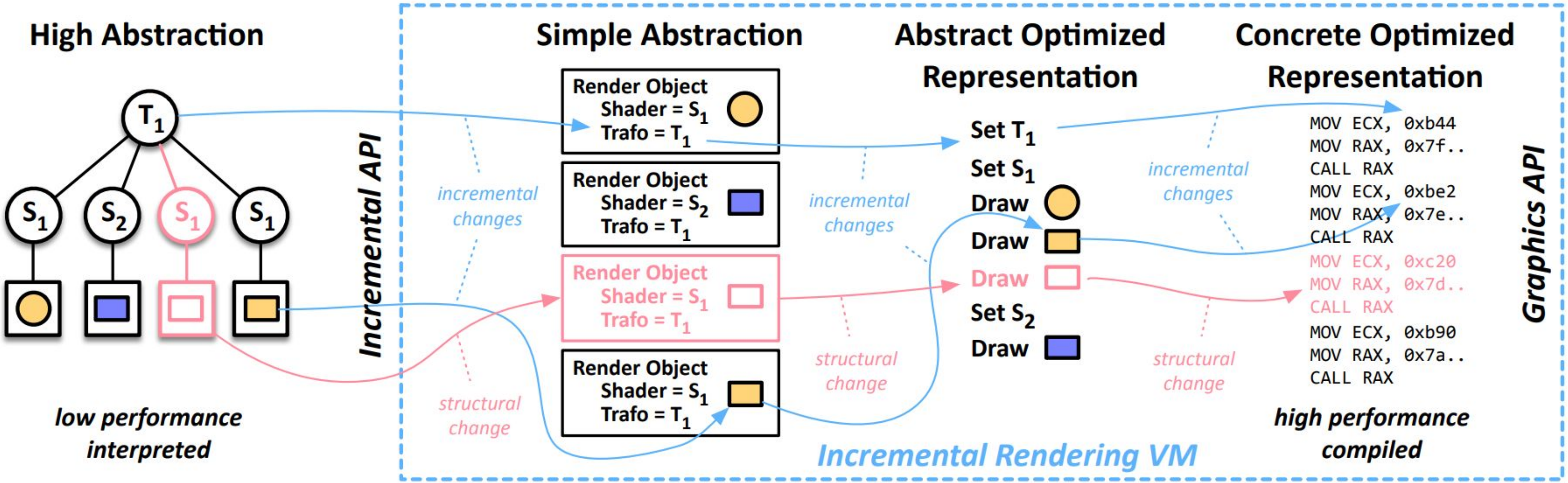
https://memegenerator.net/img/images/15955402/hysterical-raisins.jpg

An incremental renderer

- Rendering engine
  - Maps scene representation to images
- First step
  - Adaptive scene description instead of Mutable/Immutable data

```
class RenderObject {
     Shader[]            Shaders;
     IMod<BlendMode>     BlendMode;
     IMod<DrawCall>      Call;
     IMod<Array>         Vertices
     // ...
}
```





Our Implementation [Haaser et al. 2015]

First resumé

- ✔ Best possible performance
- ✔ Incremental dependency tracking
- ✔ Dependencies tell us when to render
- ✖ No functional API

Part II

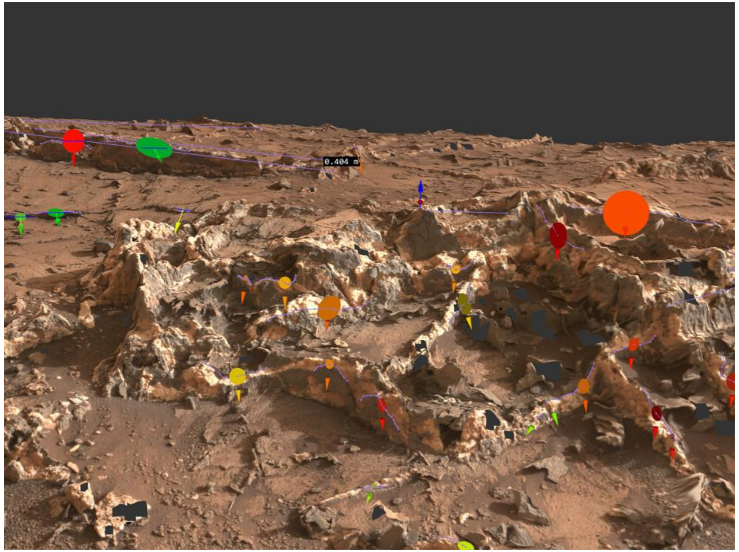
A functional Shader library

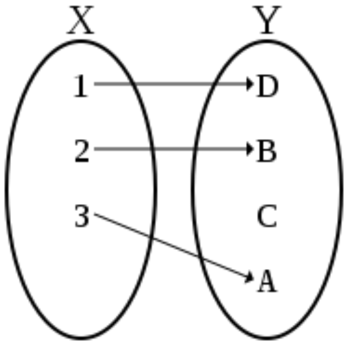
```
let skinning (v : Vertex) = // Vertex -> Quotations.Expr<Vertex> IMod<Range1d> -> IMod<MicroTime> -> ISg<'a>
    vertex {
        //let model = uniform.Bones.[uniform.MeshTrafoBone]
        let mat = getBoneTransformFrame v.vbi v.vbw
        //let mat = model * skin

        return {
            pos = mat * v.pos
            n = mat.TransformDir(v.n)
            b = mat.TransformDir(v.b)
            t = mat.TransformDir(v.t)
            vbi = V4i(-1,-1,-1,-1)
            vbw = V4d.Zero
        }
    }
```

```
vec4 SpecularColorC = texture(specularColor, fs_Diffus
vec3 vn = texture(normalMap, fs_DiffuseColorCoordinate
vec3 tn = normalize(((vn * 2.0) - vec3(1.0, 1.0, 1.0))
vec3 n = normalize(fs_Normals);
vec3 b = normalize(fs_DiffuseColorUTangents);
vec3 t = normalize(fs_DiffuseColorVTangents);
vec3 NormalsC = (((b * tn.x) + (t * tn.y)) + (n * tn.z
vec3 n1 = normalize(NormalsC);
vec3 l = normalize(fs_LightDirection);
vec3 c = normalize(fs_CameraDirection);
float diffuse = clamp(dot(n1, l), 0.0, 1.0);
float spec = clamp(dot(reflect(l, n1), (-c)), 0.0, 1.0
vec3 specc = SpecularColorC.xyz;
vec3 color = ((ColorsC.xyz * diffuse) + (specc * pow(s
ColorsOut = vec4(color, ColorsC.w);
```

Part III

Functional Programming in the Wild



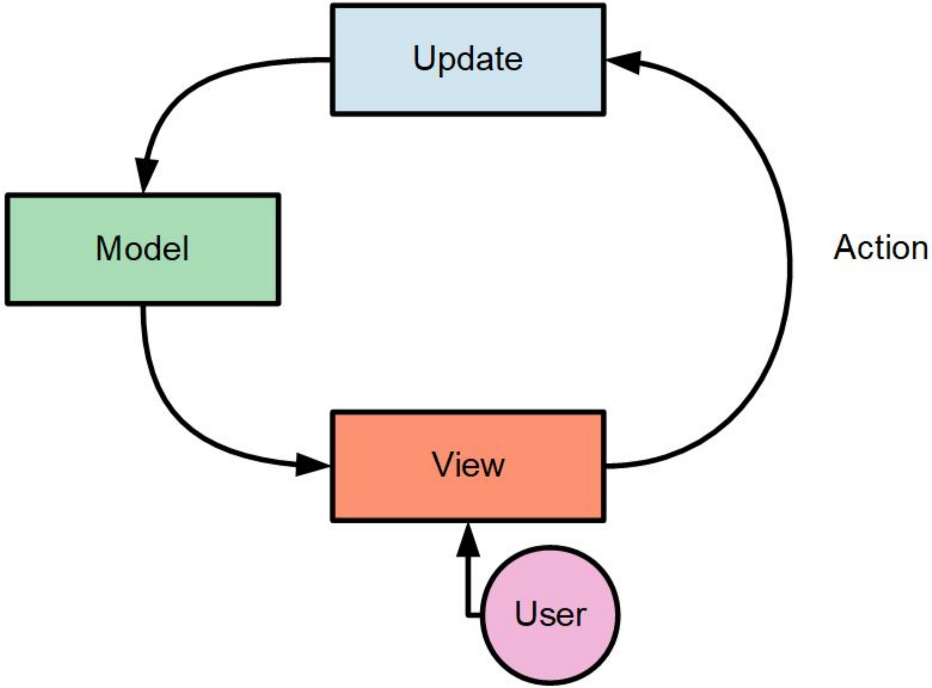


https://bar.wikipedia.org/wiki/Datei:Injection.svg

Many projects later...

- Incremental System (dependency graph) is nice
- But we still miss functional programming benefits
- Source of complexity:
  - Dealing with changes
  - Interactions
- Can we do better?

The ELM Architecture

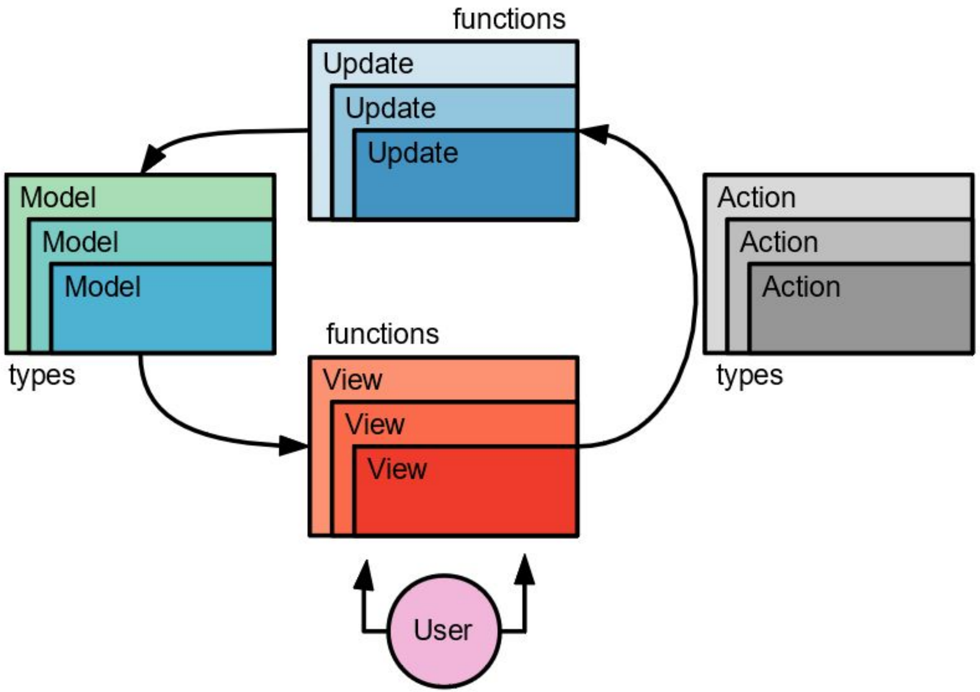


https://guide.elm-lang.org/architecture/

ELM Demo

https://ellie-app.com/yBPRbmmKvQa1

Scale through Composition



https://guide.elm-lang.org/architecture/

Immutable Data Structures

```
// Adds the specified key and value to the dictionary.
0 references
public static void Add(Dictionary<string, int> d, string key, int value)
```

```
// Returns a new map from a given map, with an additional or replaced binding.
0 references
public static Map Add(Map m, string key, int value)
```

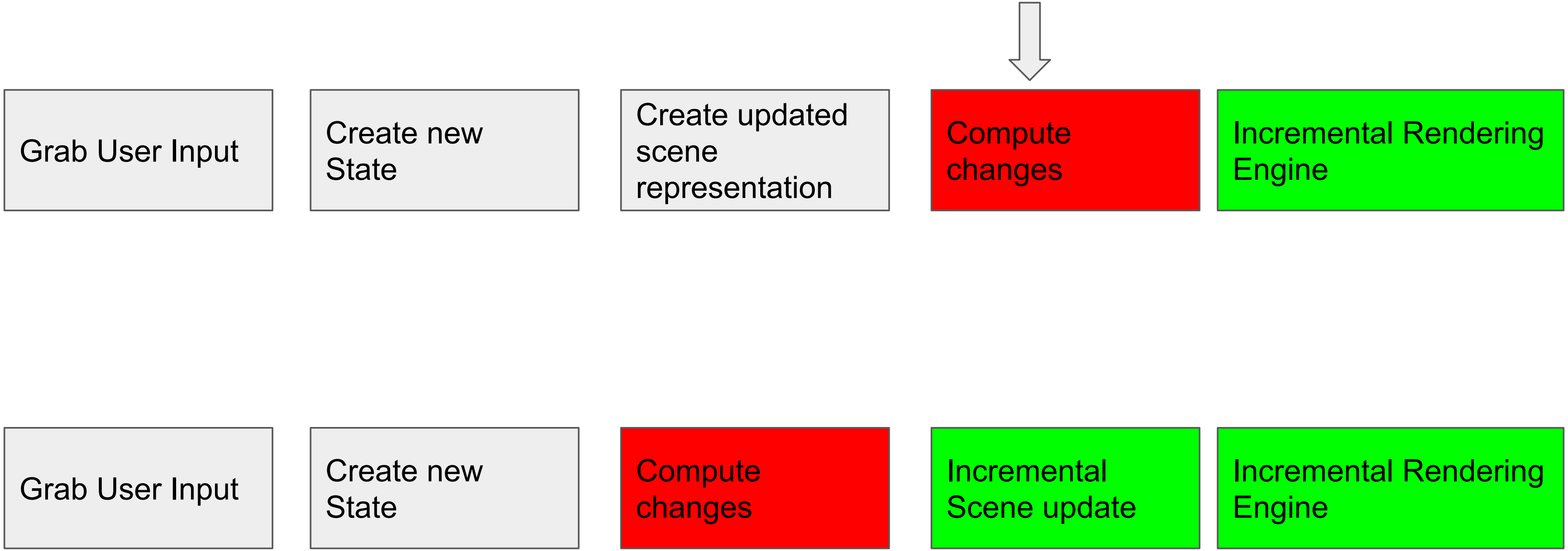
```
// Returns a new scene from a given scene, with an additional object to
// be rendered.
0 references
public static Scene AddObjToScene(Scene m, RenderableObj obj)
```

Functional Scene Representation?

- Conceptually, we get a new scene each frame

```
// Returns a new scene from a given scene, with an additional object to
// be rendered.
0 references
public static Scene AddObjToScene(Scene m, RenderableObj obj)
```

- Given a new scene, we need to extract effective changes
  - Reuse GPU resources for each scene object
- Web Frameworks extract changes at DOM level



Our approach

```
type Model =
    {
        value : int
    }

type MModel =
    {
        value : IMod<int>
    }

val applyChanges : Model -> Model -> MModel -> unit
```

recursive

```
view : Model -> Html.Html Msg
view model =
    div []
        [ button [ onClick Increment ] [ text "+" ]
        , br [] []

        , text (toString model)
        , br [] []
        , button [ onClick Decrement ] [ text "-" ]
        ]
```

```
view : MModel -> Html.Html Msg
view model =
  div [] [
        button [onClick (fun _ -> Increment)] [text "+"]
        br []
        Incremental.text (
            m.value |> Mod.map(fun x -> toString x)
        )
        br []
        button [ onClick (fun _ -> Decrement)] [text "-"]
]
```

ELM for 3D graphics

https://github.com/aardvark-platform/gpuDayDemo

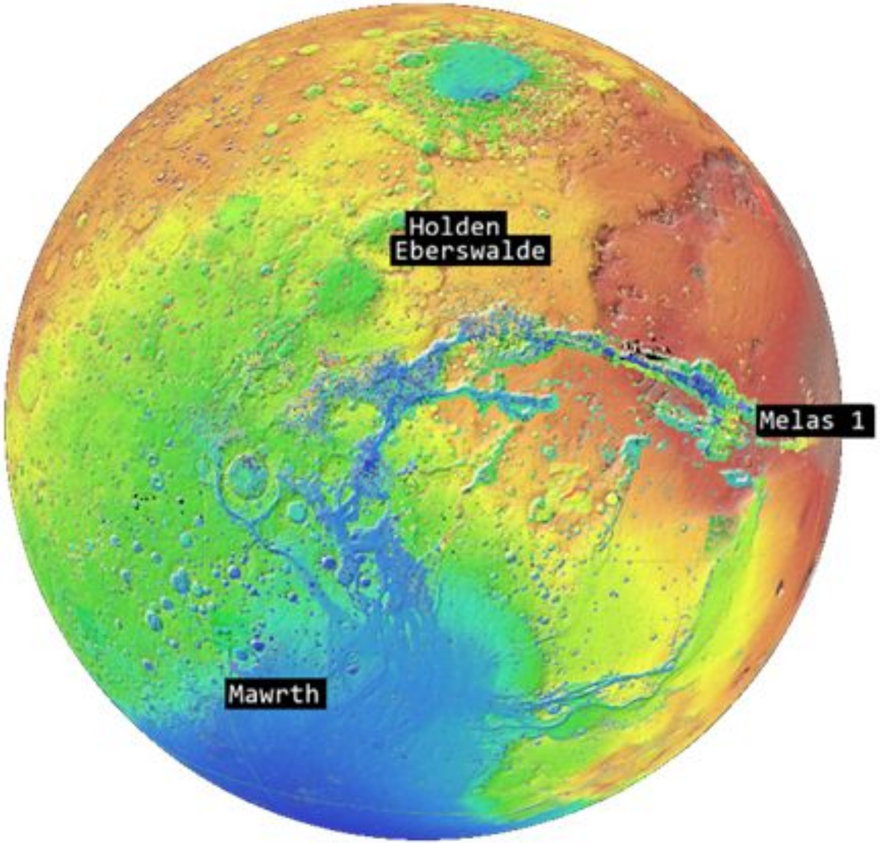
```
type Model =
  {
     finishedPolygons : list<Polygon>

     past   : Option<Model>
     future : Option<Model>
  }
```

```
let update (msg : Msg) (model : Model) =
    match msg with
      | Undo _ ->
        match m.past with
          | None -> m
          | Some p ->
              { p with future = Some m }
     | …..
```

PRo3D Viewer

3D Visualization tool for interactive visualization and analysis of the Martian surface

- Large amount of data
- Out of core asynchronous rendering
- Lots of different interactions and use cases
- Research

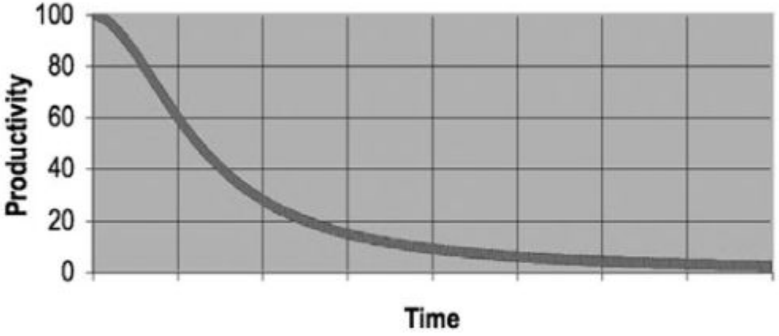


http://pro3d.space/

“The total cost of owning a mess”*

- 6 Years Development written in C# and WPF (OOP)
- “Maintenance Deadlock” - Clean code and regular refactoring?
- Out-of-Date technology and architecture
- **Functional Rewrite** (F# and HTML5)

*



*Robert C. Martin (2008) Clean Code - A Handbook of Agile Software Craftmanship. p.4-13

What can we expect from such a rewrite?

- F#
  - Functional principles enforce cleaner code by preventing side effects
  - Better testable and therefore easier to refactor

- HTML5 GUI
  - Easy throw away GUI code
  - Complex GUI elements through composition

- Additional Efforts
  - Rewrites take time
  - FP training for all members

Conclusion

- Low level performance tweaks
- High level functional programming via
  - Compilers
  - Domain specific languages
- Functional rewrite showed advantages of FP
- ELM appears to be an architecture that scales

Find us on https://aardvark.graphics