Graphics Programming In Clojure

양승헌

소개

- A Gamedev
- Emacs
- C / Lisp

Outline

- Graphics/Game programming in Clojure
- Raytracer
- Shadertone

Graphics/Game In Lisp
구글링

- 장점
  - Immutable persistant data structure
  - awesome for game scripting
  - awesome for interactive programming

- 고려할점
  - careful for performance
    - natural GC-heavy process
    - mutability

어느 게임회사 모집 공고

Naughty Dog

Game UI Scripter / Programmer

Responsibilities:

Create, modify, and debug in-game UI (menus and HU
Assist in developing and supporting both pre-existing a
Collaborate creatively with other internal departments
Assist in documenting the studio's UI tools and API
Additional responsibilities may be assigned as needed

Requirements & Skills:

Proficient in and practical experience with at least one
Proficient in and practical experience with at least one
Rapid learner, able to absorb and understand complex
Good interpersonal communication skills
Passion for games and game technology
Comfortable working closely with Designers, Artists, ar n a highly iterative environment

Bonus Skills:

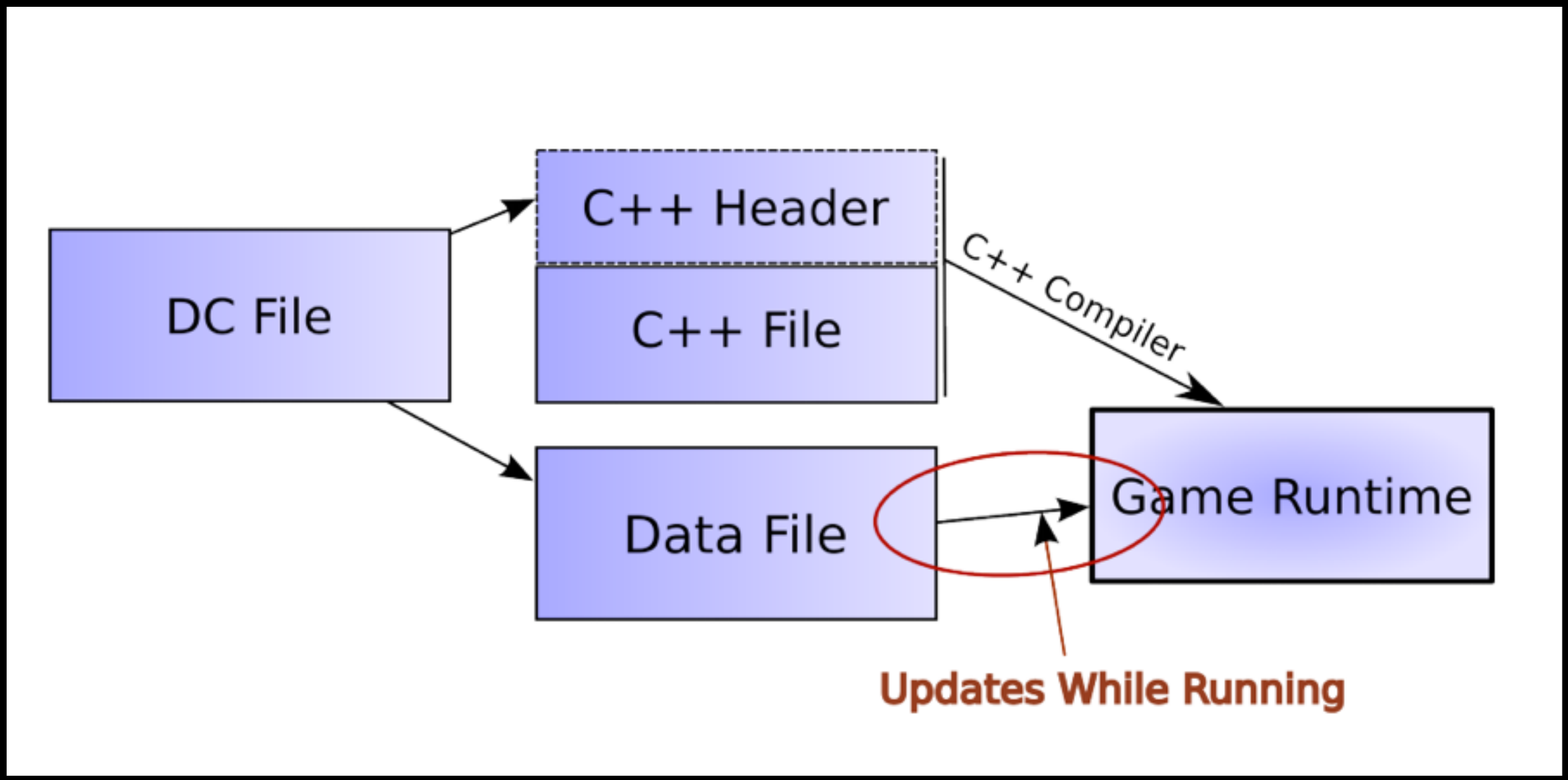
B.S. in Computer Science, Applied Mathematics or Engineering
Expertise in Flash and ActionScript 3.0 for web or UI content creation
Experience with Photoshop, Illustrator

Some exposure to Lisp-like languages (e.g., Common Lisp, Scheme/Racket, Clojure)

Apply at **jobs@naughtydog.com**

DC in Naughty Dog

Architecture of Data Compiler



https://con.racket-lang.org/2013/danl-slides.pdf

Lessons in DC

- Racket power, library support a big win
- Syntax transformation source location and performance hindered us
- S-expression based language a tough sell to industry programmers, as well as designers, and non-technical types
  - ...especially when paired up with Emacs as the editing platform.
  - Although once learnt many programmers and designers were expand and extend the language effectively
- Functional nature of the system is a big win, allowing data to be flexibly transformed to just the right runtime representation

https://con.racket-lang.org/2013/danl-slides.pdf

리습과 나

리습을 공부해도 쓸일이 없다

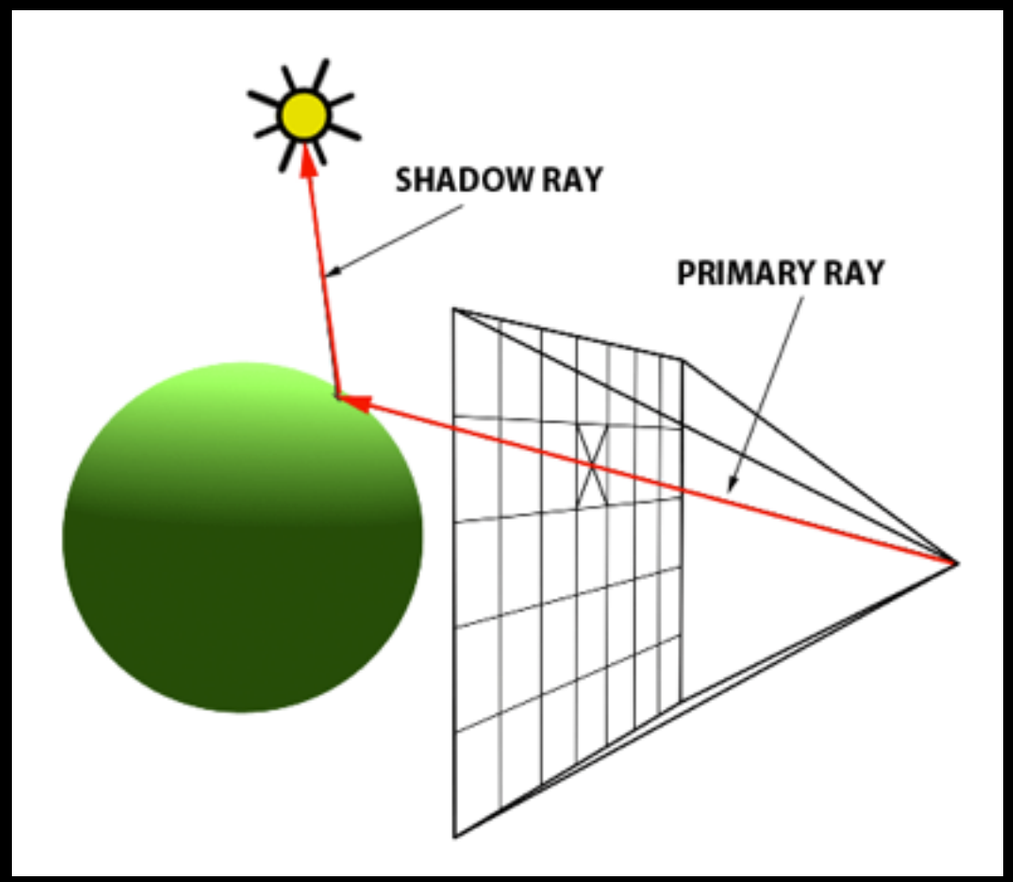
- Clojure SNG 서버를 만들어 보았다
  - 프로토타이 단계에서 필요한 기능을 금방 만들었다.
  - 금방 만들다 보니 금방 실증났다.
  - 다른 프로그래머가 쳐다 보기도 싫어한다.
  - 차라리 Python이나 Go 로 만들지 그랬냐며 빈정댄다.

- 다른 걸 해보자.
  - 굳이 Clojure로 하지 않아도 될것들
  - 그냥 최근에 C로 짜본건들
  - Raytracer / Raymarching

Raytracer

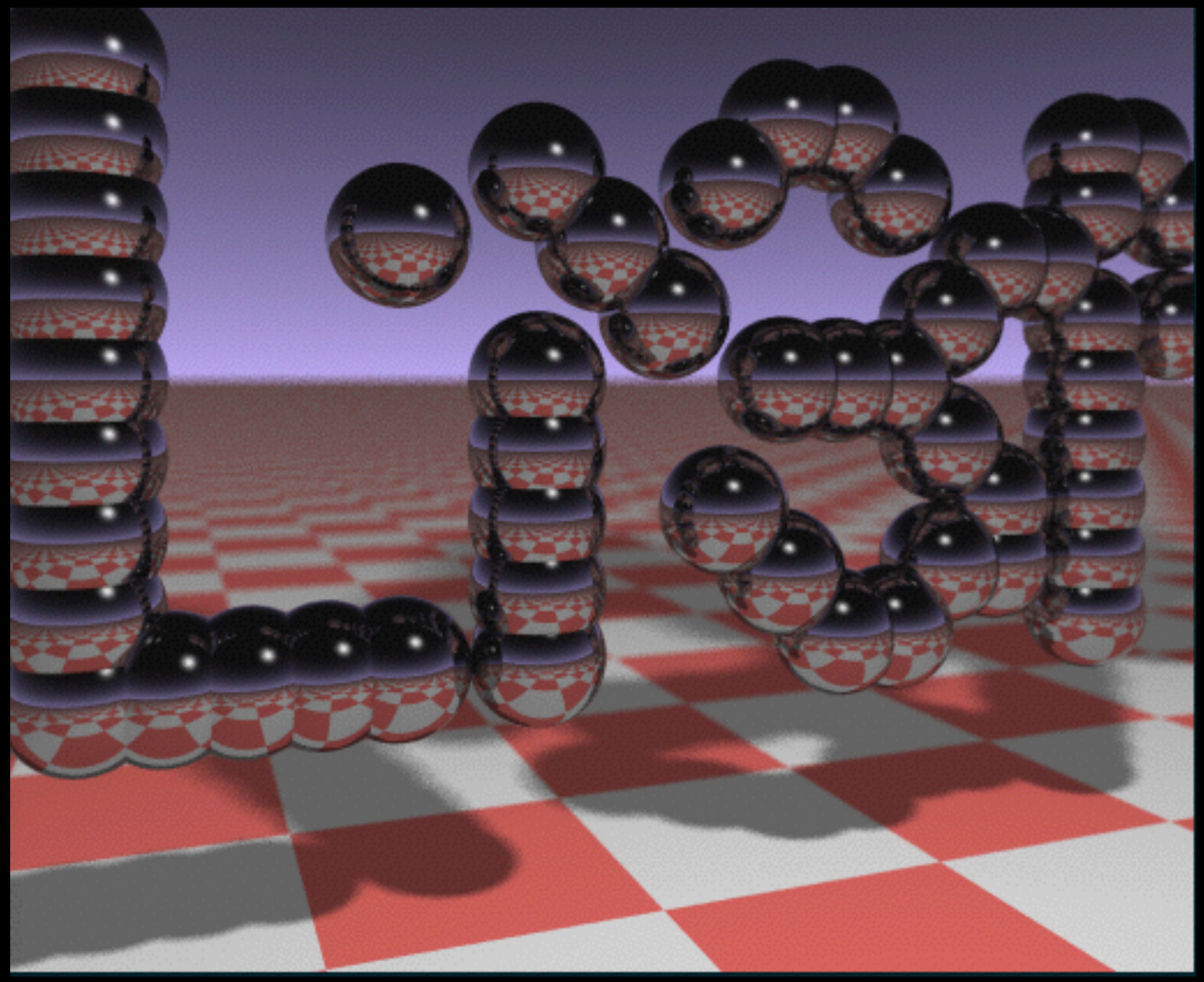
Tracing Rays :-)

- The “Hello World” in Graphics
- A numerical application(Linear Algebra)
- Rendering algorithms
- Takes lots of time



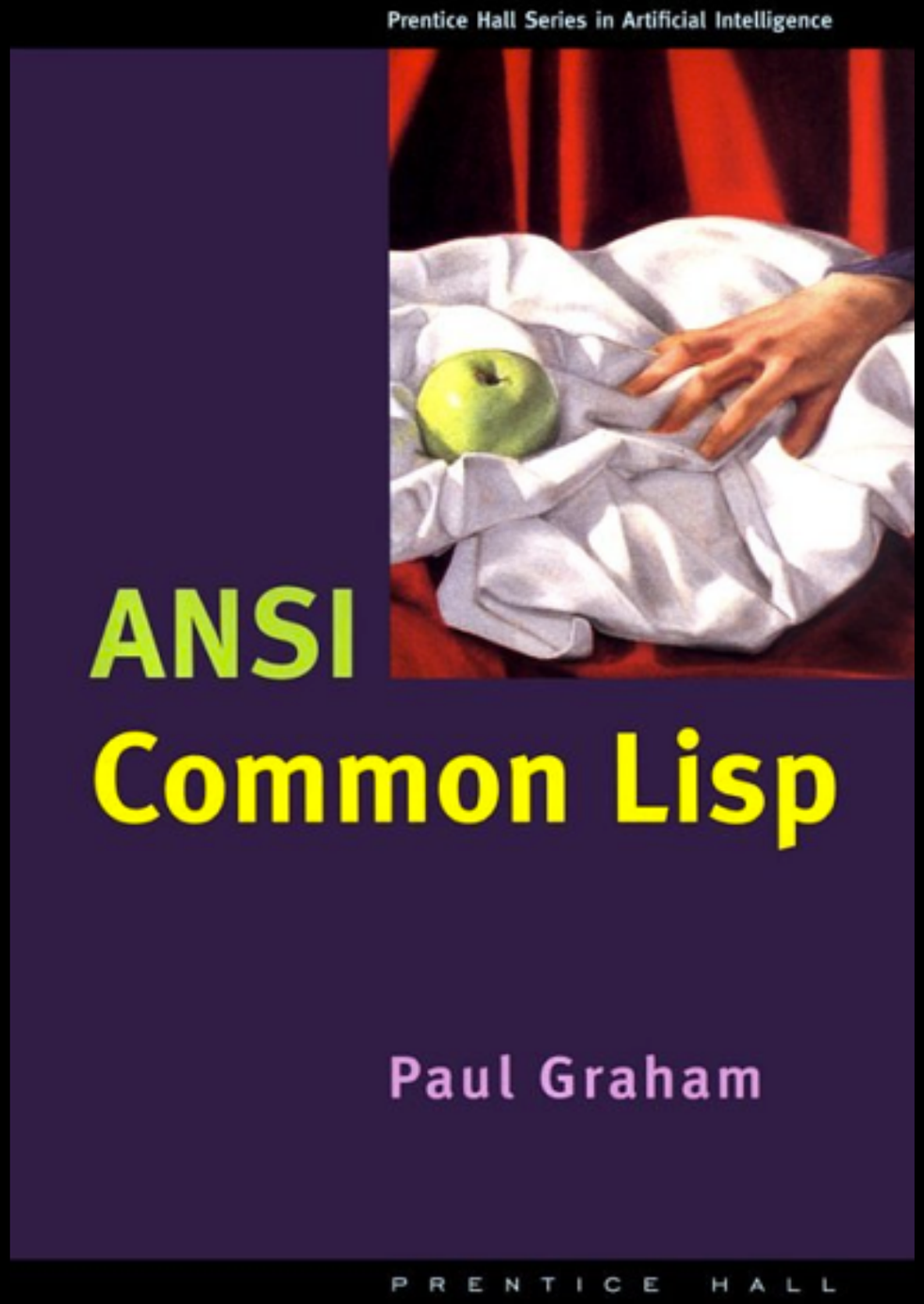
Raytracer

대략 이런 결과물

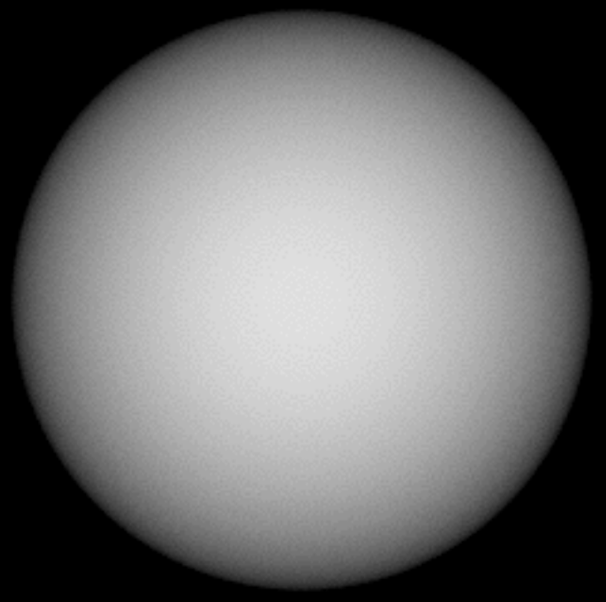


Raytracer in Common lisp

Great Paul Graham



- 9장 예제



Raytracer in Detail

Vector

ray.clj

```
(defstruct vec3 :x :y :z)

(defn make-vec3 [x y z]
  (struct vec3 x y z))

(defn sq [x]
  (* x x))

(defn magsq [vec]
  (apply + (map sq (vals vec))))

(defn mag [vec]
  (Math/sqrt (magsq vec)))

(defn unit-vector [vec]
  (let [m (mag vec)]
    (apply make-vec3 (map #(/ % m) (vals vec)))))

(defn subtract [vec1 vec2]
  (apply make-vec3 (map #(- %1 %2) (vals vec1) (vals vec2))))

(defn distance [vec1 vec2]
  (mag (subtract vec1 vec2)))
```

-:--- ray.clj 2% (26,29) (Clojure MRev) 10:35AM 1.20

Raytracer in Detail

Sphere

ray.clj

```
(defstruct sphere :center :radius)

(defn make-sphere [c r]
  (struct sphere c r))

(defn sphere-normal [s p]
  (unit-vector (subtract (:center s) p)))

(defn sphere-intersect [s p ray]
  (let [c (:center s)
        a (magsq ray)
        b (* 2 (+ (* (- (:x p) (:x c)) (:x ray))
                  (* (- (:y p) (:y c)) (:y ray))
                  (* (- (:z p) (:z c)) (:z ray))))
        c (+ (sq (- (:x p) (:x c)))
             (sq (- (:y p) (:y c)))
             (sq (- (:z p) (:z c)))
             (- (sq (:radius s))))
        n (minroot a b c)]
    (if n
      (make-vec3 (+ (:x p) (* n (:x ray)))
                 (+ (:y p) (* n (:y ray)))
                 (+ (:z p) (* n (:z ray)))))))
```

-:--- ray.clj 27% (59,46) (Clojure MRev) 10:36AM 1.56

Raytracer in Detail

Tracing

ray.clj

```
(defn lambert [s int ray]
  (let [norm (sphere-normal s int)]
    (max 0 (+ (* (:x ray) (:x norm))
              (* (:y ray) (:y norm))
              (* (:z ray) (:z norm))))))

(defn first-hit [world p ray]
  (->> (reduce (fn[h v]
                 (if-let [i (sphere-intersect v p ray)]
                   (conj h [i v]) h)) [] world)
       (sort-by #(distance (first %) p))
       first))

(defn send-ray [w src ray]
  (if-let [[loc obj] (first-hit w src ray)]
    (* (lambert obj loc ray) 0.85)
    0))

(defn color-at [w eye x y]
  (let [ray (unit-vector (subtract (make-vec3 x y 0) eye))]
    (send-ray w eye ray)))

(defn ray-trace-file [world eye w h]
```

-:--- ray.clj 51% (82,0) (Clojure MRev) 10:37AM 1.29

Raytracer in Detail

Writing Numbers into file

ray.clj

```
(let [uv (for [x (range 0 w) y (range 0 h)] [x y])
      colors (pmap #(let [[x y] %]
                      [x y (color-at world eye x y)]) uv)]
  (with-open [wrtr (io/writer "paul.ppm")]
    (.write wrtr (str "P3\n" w " " h "\n255\n"))
    (doseq [[x y c] colors]
      (let [col (int (* c 255.99))]
        (.write wrtr (format "%d %d %d\n" col col col))))))
```

-:**- ray.clj 74% (106,0) (Clojure cider MRev) 3:19PM 1.34

Raytracer in Detail

PPM file

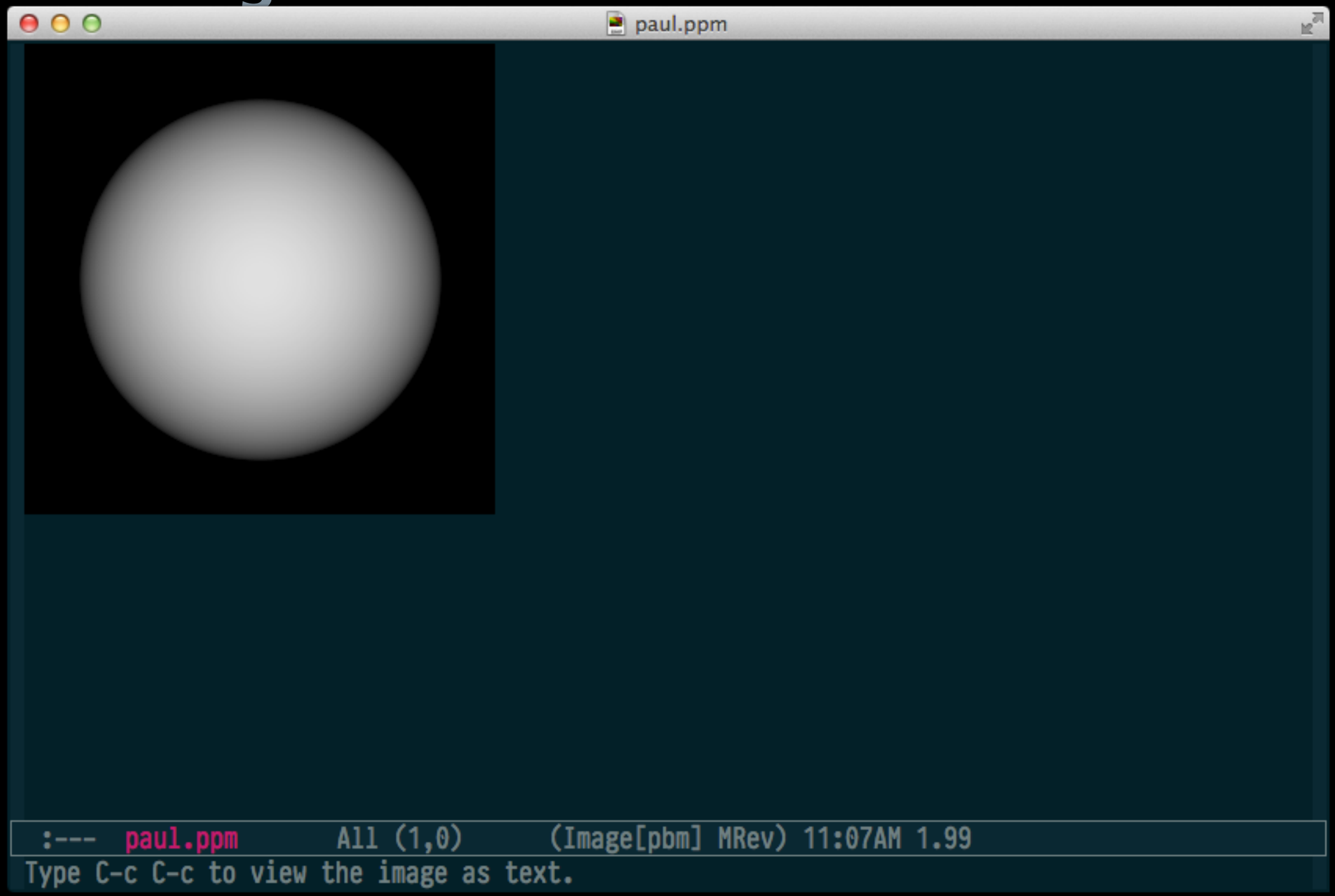
ray.ppm

```
P3
400 200
255
206 227 255
206 227 255
205 227 255
205 227 255
205 227 255
205 227 255
205 227 255
205 227 255
205 227 255
205 227 255
205 227 255
205 226 255
205 226 255
205 226 255
205 226 255
205 226 255
205 226 255
205 226 255
205 226 255
204 226 255
```

```
 :---  ray.ppm        Top (1,0)     (Fundamental Image[pbm] MRev) 11:06AM 2.57
Type C-c C-c to view the image as an image.
```

Raytracer in Detail

Viewing PPM file



Advanced Raytracer

Features

- Shadow
- Lambertian Material
- Metal Material
- Dielectric Material
- Camera
- Performance

clojore.core.matrix

Vector[1d] / Matrix[md] library

core.clj

```
(ns oneweek-clj.core
  (:require [clojure.core.matrix :as mat]
  (:gen-class))

(mat/set-current-implementation :vectorz)

(defn vec3
  [u v w]
  (mat/matrix [u v w]))

(defn ray
  [a b]
  {:origin a :direction b})

(defn reflect
  [v n]
  (mat/sub v (mat/mul 2.0 (mat/dot v n) n)))

(defn refract
  [v n ni-over-nt]
  (let [uv (mat/normalise v)
        dt (mat/dot uv n)
        discriminant (- 1.0 (* ni-over-nt ni-over-nt (- 1 (* dt dt))))]
    (if (> discriminant 0)
```

-:**- core.clj Top (20,0) (Clojure MRev) 10:50AM 1.32

Auto-saving...done

defstruct vs defrecord

Why should I use defrecord in clojure

- Performance
- With additional macro machinery around defrecord, I can get field validation, default values, and whatever other stuff I want
- Records can implement arbitrary interfaces or protocols (maps can't)
- Records act as maps for most purposes
- keys and vals return results in stable (per-creation) order

Materials in defrecord

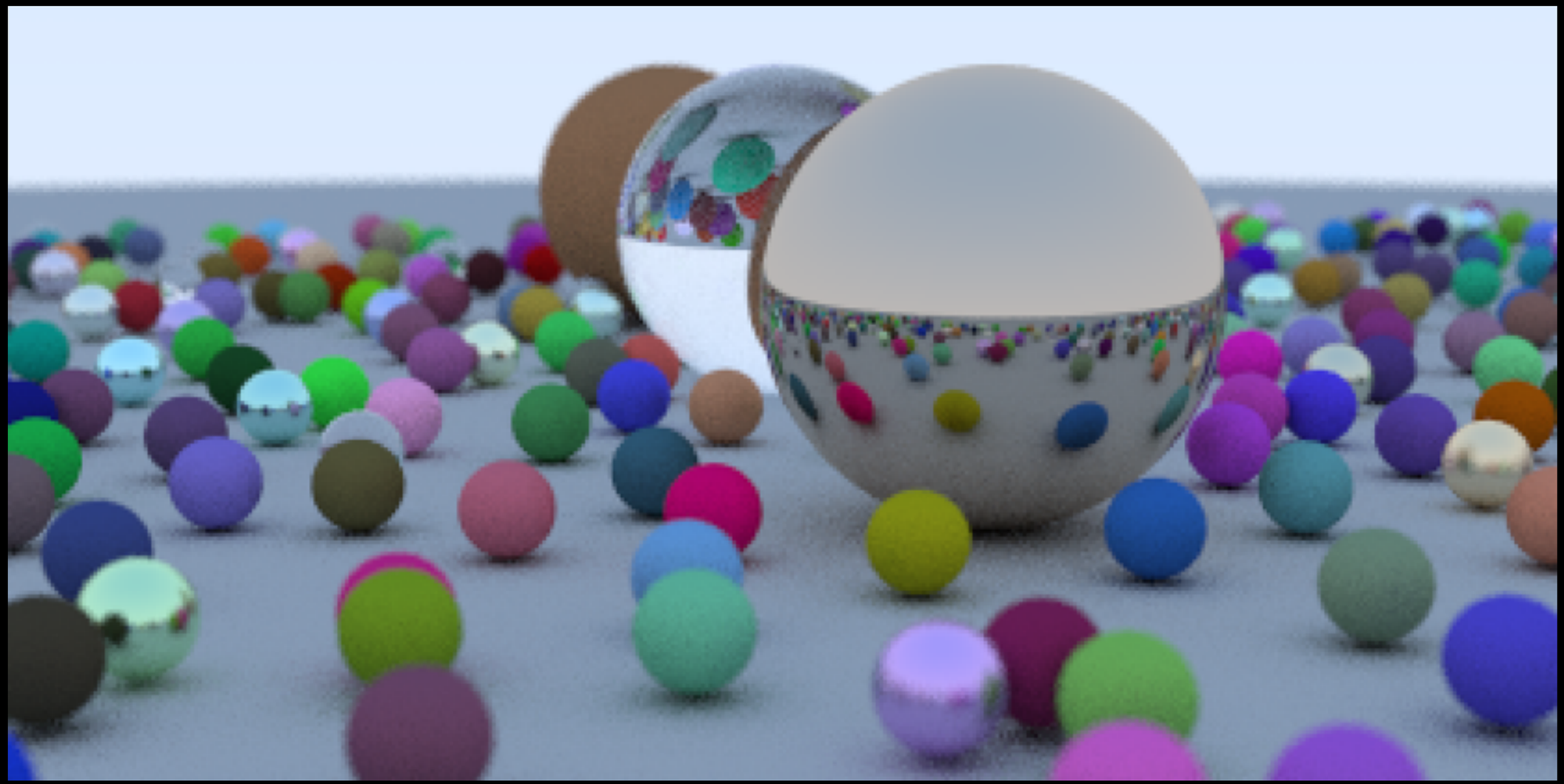
core.clj

```clojure
(defprotocol shader
  (scatter [this ray-in hrec]))

(defrecord lambertian [albedo]
  shader
  (scatter [this ray-in {:keys [t p normal material]}]
    (let [target (mat/add p normal (rand-in-unit-sphere))]
      {:scattered (ray p (mat/sub target p))
       :attenuation albedo})))

(defrecord metal [albedo fuzz]
  shader
  (scatter [this ray-in {:keys [t p normal material]}]
    (let [reflected (reflect (mat/normalise (:direction ray-in)) normal)
          scattered (ray p (mat/add reflected
                                    (mat/mul fuzz
                                             (rand-in-unit-sphere))))]
      (if (> (mat/dot (:direction scattered) normal) 0)
        {:scattered scattered
         :attenuation albedo}))))

(defn schlick
  [cosine ri]
  (let [r0 (/ (- 1.0 ri) (+ 1.0 ri))
```

-:**- core.clj 21% (108,0) (Clojure MRev) 10:53AM 0.98

Auto-saving...done

Output



Using GPU in Clojure

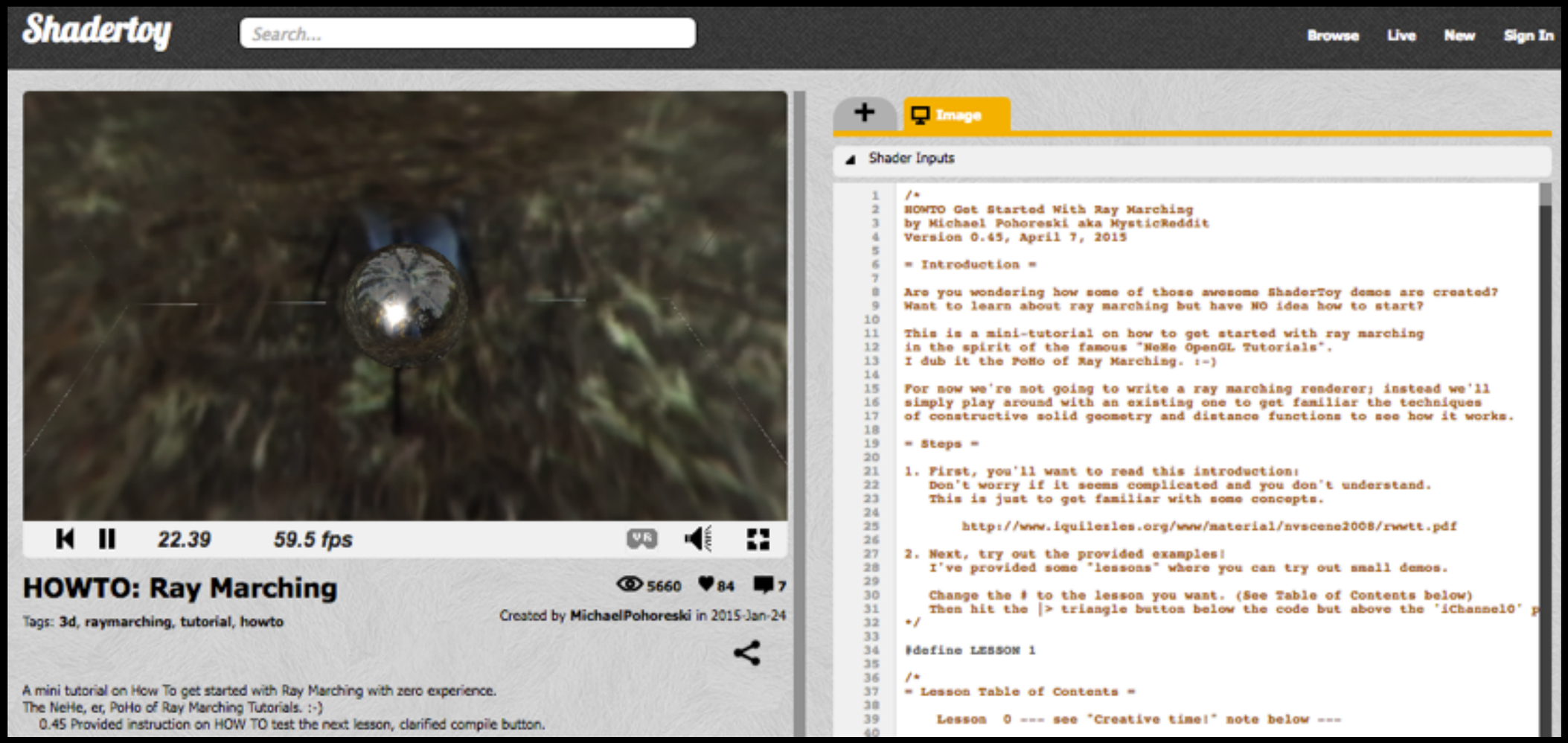
GPU programming libraries

- Java Graphics API
- Penumbra(OpenGL Wrapper)
- Calx(OpenCL Wrapper)
- ClojureCL(OpenCL Wrapper)
- Neanderthal(BLAS Wrapper)
- Shadertone(Shader programming)

Shadertone

Shadertoy in Clojure

- shadertoy.com



Shadertone

Usage

core.clj<sot>

```
(ns sot.core
  (:use [overtone.live])
  (:require [shadertone.tone :as t])
  (:gen-class))

(defn -main [& args]
  ;; start up shadertone
  (t/start "src/sot/throb.glsl")

  ;; wait around...
  (Thread/sleep (* 10 1000))

  ;; Stop shadertone
  (t/stop)
  (println "Done. The program should stop.")
  (System/exit 0))
```

```
-:**-  core.clj<sot>   All (13,5)    Git-master  (Clojure MRev) 11:15AM 1.27
```

Shadertone

GLSL(OpenGL Shading Language)

throb.glsl

```
uniform float iOvertoneVolume;
//uniform float iGlobalTime;
void main(void) {
    gl_FragColor = vec4(cos(iGlobalTime) * 0.5 + 0.5,
                        0.5,
                        0.5,
                        1.0);
}
```

-:--- throb.glsl   All (1,0)   Git:master (GLSL hs ElDoc MRev Abbrev Fill) 11:16AM 1.3

Shadertone

GLSL(raymarching)

raymarching1.glsl

```
void main(void)
{
    vec2 window_pos        = -1.0 + 2.0*(gl_FragCoord.xy/iResolution.xy);
    vec3 camera_up         = vec3(0,1,0);
    vec3 camera_lookat     = vec3(0,0,0);
    vec3 camera_pos        = vec3(5.0*cos(0.1*iGlobalTime),
                                  5.0,
                                  5.0*sin(0.1*iGlobalTime));
    vec3 light_pos         = vec3(5.0,10.0,5.0);
    vec3 norm_camera_dir   = normalize(camera_lookat-camera_pos);
    vec3 u                 = normalize(cross(camera_up,norm_camera_dir));
    vec3 v                 = cross(norm_camera_dir,u);
    vec3 near_plane_center = (camera_pos+norm_camera_dir);
    vec3 near_plane_coord  = (near_plane_center +
                              window_pos.x*u*iResolution.x/iResolution.y +
                              window_pos.y*v);
    vec3 norm_eye_ray      = normalize(near_plane_coord-camera_pos);

    const float max_depth  = 100.0;
    vec2 dist_id           = vec2(0.02,0.0);
    float cur_depth        = 1.0;
    vec3 cur_color,p;

    // ray-march to find the depth and object intersected
```

-:--- raymarching1.glsl 55% (115,0) (GLSL hs ElDoc MRev Abbrev Fill) 11:18AM 1.28

Shadertone

Lisp-like GLSL

03demo_translate.clj

```
(ns demo3
  (:require [shadertone.shader :as s]
            [shadertone.translate :as trans]))

;; translate this Clojure-like code to a GLSL shader
(trans/defshader simple
  '((uniform vec3  iResolution)
    (uniform float iGlobalTime)
    (defn void main []
      (setq vec2 uv (/ gl_FragCoord.xy iResolution.xy))
      (setq float b (abs (sin iGlobalTime)))
      (setq gl_FragColor (vec4 uv.x uv.y b 1.0)))))
;;(print simple)

;; wrap the shader in an atom to allow it to be watched & modified
(def simple-atom (atom simple))
(s/start simple-atom)

;; swap it out for something simple
(swap! simple-atom (fn [x] "
void main(void) {
  gl_FragColor = vec4(1.0,0.5,0.25,1.0);
}
"))
```

-:--- 03demo_translate.clj   Top (1,0)   (Clojure MRev) 11:19AM 1.29

Lessons

- Immutable vs mutable의 특성을 이해하고 적재적소에 활용하자
- defstruct 보다는 defrecord
- GPU를 활용할 수 있으나 아직 좀 부족한 감이 있다
- CPU최적화(e.g. SIMD) 방법은 찾지 못했다.(VM?)