毕业设计组会报告

徐海洋

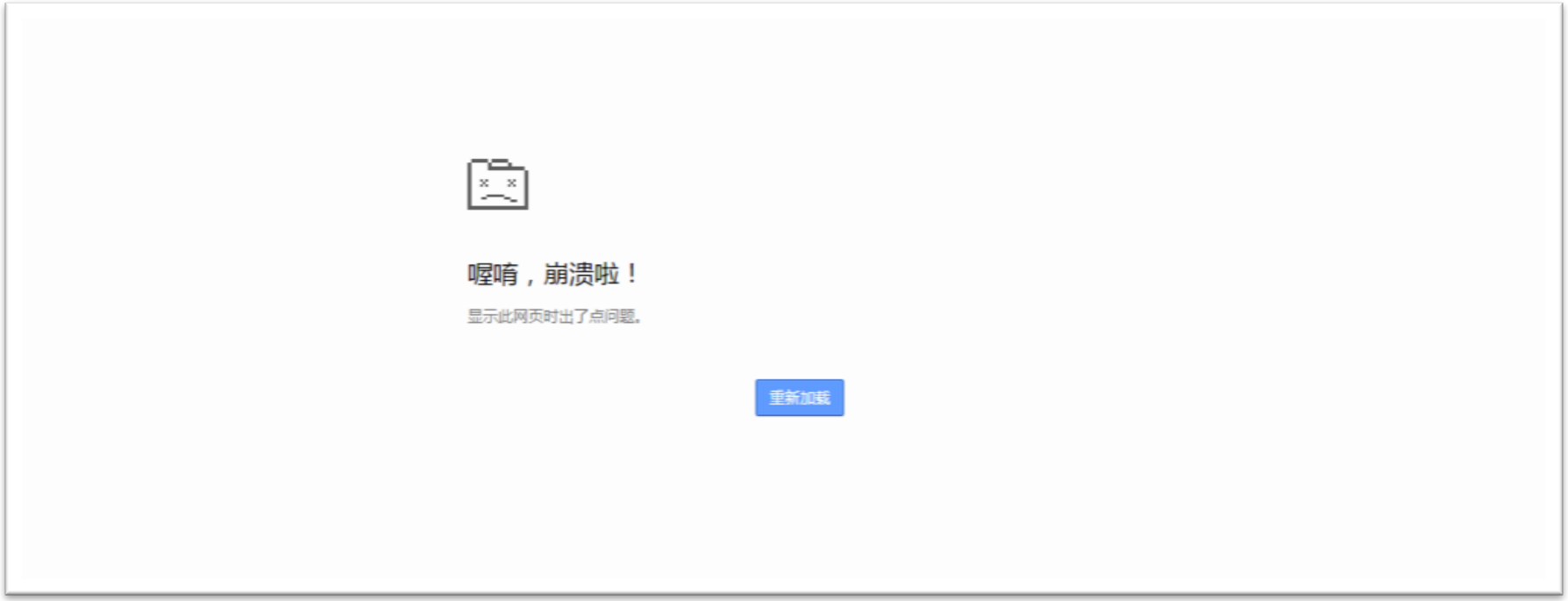
目录

项目概述

基于 WebGL 利用 GPU 加速的前端图像处理框架

动机

- 在一个医疗图像标注系统的项目中，高分辨率的图像在前端直接使用 CPU 进行像素处理会造成浏览器崩溃。



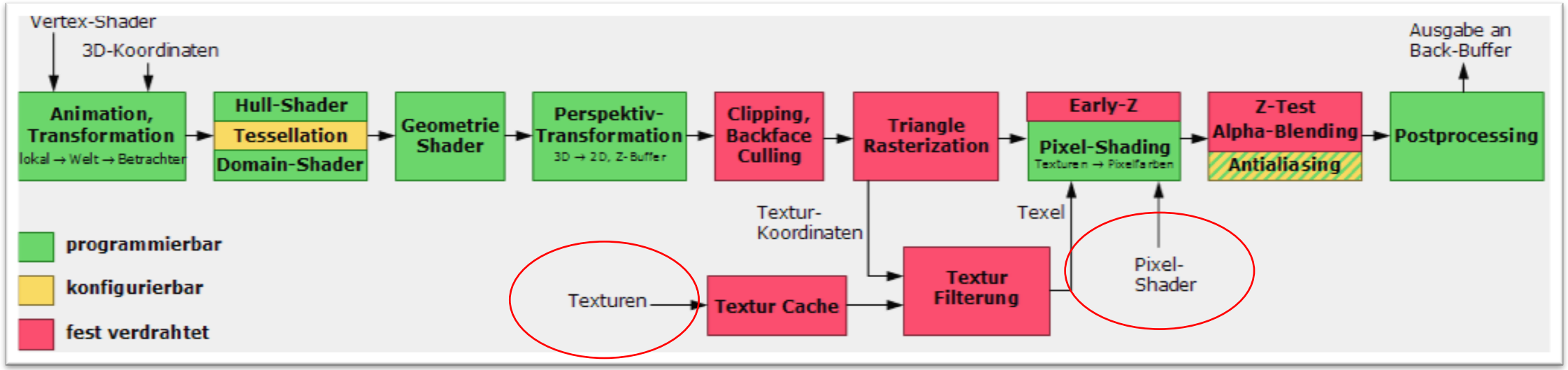
目标

- 希望前端工作与所学习的图像处理、计算机图形学的知识相融合。
- 实现一个高效、易用、可拓展的框架。

调研的收获

关于技术原理、框架设计等方面

渲染管线



Graphics pipeline - Wikipedia

着色器

JS HTML CSS Codepen JSFiddle Result Run WebGL2Fundamentals

```
36 var fragmentShaderSource = `#version 300 es
37
38 // fragment shaders don't have a default precision so we need
39 // to pick one. mediump is a good default. It means "medium precision"
40 precision mediump float;
41
42 // our texture
43 uniform sampler2D u_image;
44
45 // the texCoords passed in from the vertex shader.
46 in vec2 v_texCoord;
47
48 // we need to declare an output for the fragment shader
49 out vec4 outColor;
50
51 void main() {
52   outColor = texture(u_image, v_texCoord).rgba;
53 }
54 `;
55
56 var image = new Image();
57   image.src = "https://webgl2fundamentals.org/webgl/resources/leaves.jpg";  // MUST BE SAME DOMAIN
58   image.onload = function() {
59   render(image);
60 };
```

WebGL2 Image Processing – WebGL2Fundamentals

着色器

JS HTML CSS Codepen JSFiddle Result Run WebGL2Fundamentals

```
36 var fragmentShaderSource = `#version 300 es
37
38 // fragment shaders don't have a default precision so we need
39 // to pick one. mediump is a good default. It means "medium precision"
40 precision mediump float;
41
42 // our texture
43 uniform sampler2D u_image;
44
45 // the texCoords passed in from the vertex shader.
46 in vec2 v_texCoord;
47
48 // we need to declare an output for the fragment shader
49 out vec4 outColor;
50
51 void main() {
52   outColor = texture(u_image, v_texCoord).bgra;
53 }
54 `;
55
56 var image = new Image();
57   image.src = "https://webgl2fundamentals.org/webgl/resources/leaves.jpg";  // MUST BE SAME DOMAIN
58   image.onload = function() {
59   render(image);
60 };
```

WebGL2 Image Processing – WebGL2Fundamentals

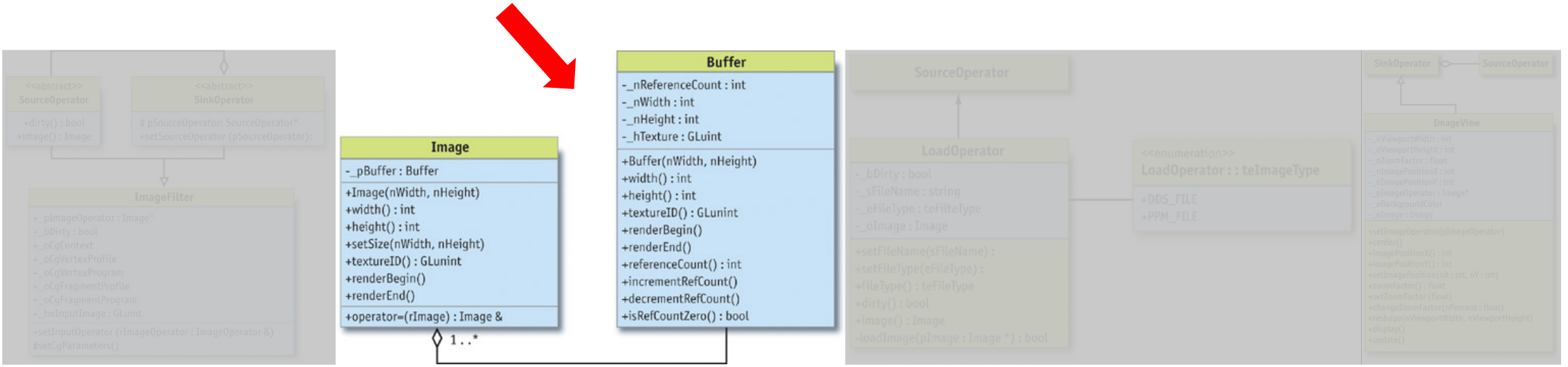
纹理

```
function initTextures() {
  cubeTexture = gl.createTexture();
  cubeImage = new Image();
  cubeImage.onload = function() { handleTextureLoaded(cubeImage, cubeTexture); }
  cubeImage.src = "cubetexture.png";
}

function handleTextureLoaded(image, texture) {
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);
  gl.generateMipmap(gl.TEXTURE_2D);
  gl.bindTexture(gl.TEXTURE_2D, null);
}
```

Using textures in WebGL - MDN

框架



GPU Gems - NVIDIA Developer

易用

Usage

Add a canvas element to your HTML with an id:

```
<canvas id="viewerCanvas" width="500" height="300"></canvas>
```

Then initialize the viewer in javascript:

```
// start a simple image viewer mit zooming and dragging features
var myImageViewer = new ImageViewer('viewerCanvas', 'myImage.jpg');
```

That's it!

Creating the viewer

```
function ImageViewer(canvasId, imageUrl, options)
```

HTML5/Canvas Image Viewer - Github

一些问题

在试验性工作中

环境

- 图片跨域访问：利用 webpack-dev-sever 构建本地测试服务器
- 着色器的相关矩阵和向量计算：依赖 gl-matrix 的函数库

DEMO

File: **a.js**

```
var f = Fundus(canvasId, imageUrl);
var b = document.querySelector('#btn');
b.onclick = function () {
  f.gray();
};
```

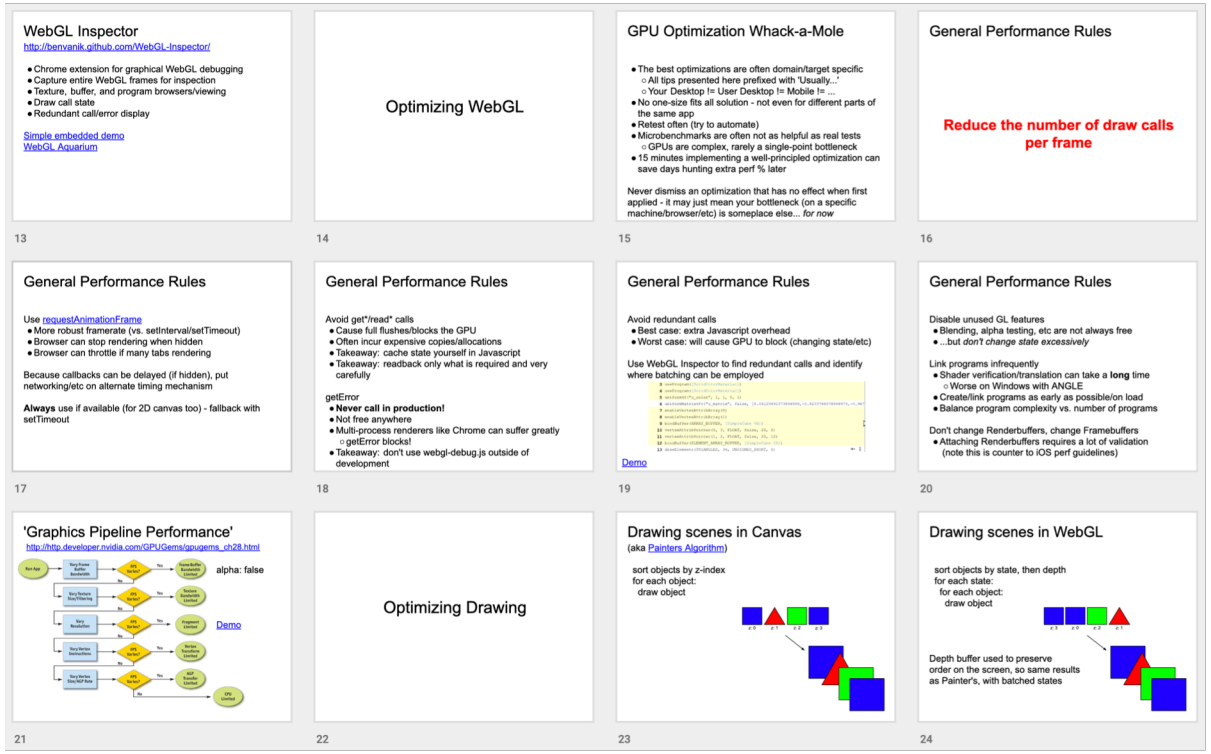
问题

探索

- 严重的问题：采用纹理渲染方法后，图像的分辨率被压缩
- 可能原因：原图作为纹理直接绑定至较小的画布时，默认的采样方式导致失真（内优化）
- 解决方案：使用顶点着色器缩放（尚未试验）

调试与优化

- *Ben Vanik & Ken Russell* in New Game 2011 SF
- Debugging and Optimizing WebGL Applications



寒假的工作

目标及安排

目标

- 高效：
  - 工程细节优化，非图像处理算法部分
  - 探索渲染效率评价标准或方式
- 易用：
  - 继续完善基本的框架模型
  - 添加对图像的基本功能，从显示到滤镜、缩放

可拓展性

- 如何让库真正成为框架？
- 如何提供一种更好的着色器编写或引用的方式？

附录

相关资料

清单

- Graphics pipeline - Wikipedia
- WebGL2 Image Processing – WebGL2Fundamentals
- Using textures in WebGL – MDN
- GPU Gems - NVIDIA Developer
- HTML5/Canvas Image Viewer - Github
- DevServer – webpack
- glMatrix - Github

参考项目

evanw / webgl-filter ★ Unstar

An image editor in WebGL

JavaScript ★ 755 107 Updated on 14 Oct 2016

phoboslab / WebGLImageFilter ★ Unstar

Fast image filters for Browsers with WebGL support

JavaScript ★ 190 41 Updated on 4 Nov 2015

evanw / glfx.js ★ Unstar

An image effects library for JavaScript using WebGL

JavaScript ★ 2,111 293 Updated on 21 Mar 2016

gre / gl-react ★ Unstar

gl-react – React library to write and compose WebGL shaders

JavaScript ★ 1,713 93 Updated on 7 Nov

gpujs / gpu.js ★ Unstar

GPU Accelerated JavaScript

JavaScript ★ 6,639 315 Updated 13 days ago

npm v2.2.8 downloads 7k/m license MIT build error coverage 66%

Cornerstone Core

Greenkeeper enabled

Cornerstone is an open source project with a goal to deliver a complete web based medical imaging platform. This repository contains the Cornerstone Core component which is a lightweight JavaScript library for displaying medical images in modern web browsers that support the HTML5 canvas element. Cornerstone Core is not meant to be a complete application itself, but instead a component that can be used as part of larger more complex applications. See the OHIF Viewer for an example of using the various Cornerstone libraries to build a simple study viewer.

Cornerstone Core is agnostic to the actual container used to store image pixels as well as the transport mechanism used to get the image data. In fact, Cornerstone Core itself has no ability to read/parse or load images and instead depends on one or more ImageLoaders to function.

The goal here is to avoid constraining developers to work within a single container and transport (e.g. DICOM) since images are stored in a variety of formats (including proprietary). By providing flexibility with respect to the container and transport, the highest performance image display may be obtained as no conversion to an alternate container or transport is required. It is hoped that developers feel empowered to load images from any type of image container using any kind of transport. See the CornerstoneWADOImageLoader project for an example of a DICOM WADO based Image Loader.

Cornerstone Core is agnostic to the exact interaction paradigm being used. It does not include any mouse, touch or keyboard bindings to manipulate the various image properties such as scale, translation or ww/wc. The goal here is to avoid constraining developers using this library to fit into a given ui paradigm. It is hoped that developers are empowered to create new paradigms possibly using new input mechanisms to interact with medical images (e.g. Kinect or Accelerometer. Cornerstone does provide a set of API's allowing manipulation of the image properties via javascript. See the CornerstoneTools library for an example of common tools built on top of Cornerstone.