A Serverless Convex Hull Web Application

ECE465 Final Project

Jonathan Lam

The Cooper Union for the Advancement of Science and Art

May 13, 2021

Why convex hull web app?

Why convex hull?

- Convex hull is a useful algorithm, e.g., for computer vision
- Originally planned to do HPC, convex hull is a divide-and-conquer algorithm

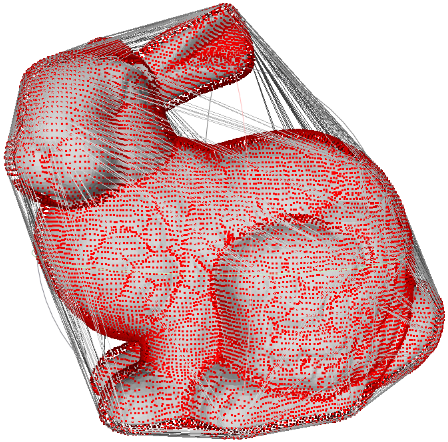
Why serverless?

- Competitive pricing to traditional servers
- Better decoupling/maintainability of services

Project trajectory

$$\begin{tikzcd}
\text{custom HPC distributed 3-D qhull algorithm} \arrow[d] \\
\text{web app using existing 3-D qhull library}
\end{tikzcd}$$

(3-D) Convex hull



(3-D) Convex hull

Implementation difficulties

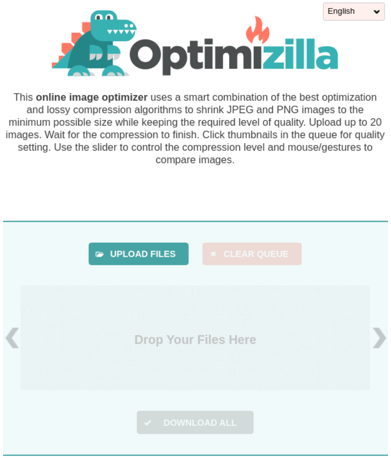
- Handle degenerate cases
- FP numerical precision issues
- Need to merge coplanar faces when they emerge to avoid problematic geometries: “thick plane” coplanar test
- Non-triangular faces due to face merging
- Newell’s method for arbitrary (planar) polygonal normals – more robust to nonplanarity
- Half-edge data structure
- DFS over visible faces to find horizon
- Have to resolve possible topological errors on face merging

(3-D) Convex hull resources

- "Implementing Quickhull – Dirk Gregorius" – an amazing visual tutorial on 3-D QuickHull
- qhull – a well-known robust QuickHull (and related algorithms) implementation
- A robust and academic QuickHull3D implementation in Java

Vision of finished product

Something like Optimizilla – one-off image processing


This **online image optimizer** uses a smart combination of the best optimization and lossy compression algorithms to shrink JPEG and PNG images to the minimum possible size while keeping the required level of quality. Upload up to 20 images. Wait for the compression to finish. Click thumbnails in the queue for quality setting. Use the slider to control the compression level and mouse/gestures to compare images.

Tech stack

```
aws-cli/2.1.32
```

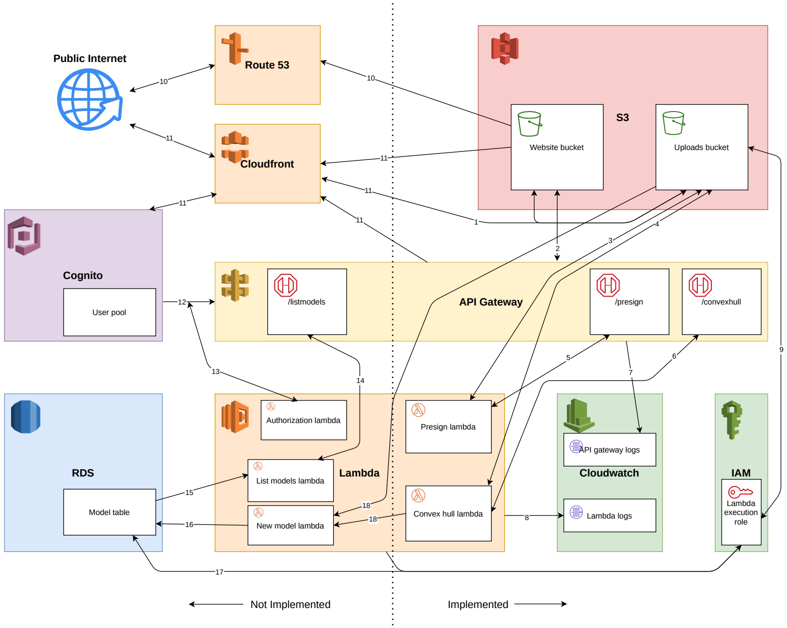
Build

```
github.com/aws/aws-lambda-go/lambda
github.com/aws/aws-sdk-go/service/s3
github.com/markus-wa/quickhull-go
```

Lambdas

Frontend

Architecture diagram



Architecture diagram notes

1. Access to uploads bucket by presigned URLs only.
2. Appopriate CORS headers for website use.
3. Presign lambda has read access to uploads bucket.
4. Convex hull lambda has GET/PUT access to uploads bucket.
5. Presign endpoint calls presign lambda.
6. Convex hull endpoint calls convex hull lambda.
7. API Gateway stage has logging set up.
8. Each lambda has logging set up (to different loggroups).
9. Lambdas need access to uploads bucket.
10. Route 53 acts as a DNS server.
11. Cloudfront acts as a CDN for the website, uploads bucket, API Gateway, and authorization page. It also supports HTTPS (necessary for Cognito).
12. Cognito authorizes API requests using the users access token.
13. Authorization lambda verifies the user's access token.
14. Listmodels endpoint calls listmodels lambda.
15. Listmodels lambda retrieves models entries from database.
16. New model lambda creates new model entry in database.
17. Listmodels/Newmodel lambdas require permissions for RDS.
18. On new upload (PUT request) or creation of a hull (a new model), newmodel lambda will trigger

Components

Front-end:

- Vue 3 framework and Vite build tool
- THREE.js for loading/display OBJ files
- Static web hosting using S3

API:

- CORS policy for front-end, lambda policy
- Endpoints: /presign, /convexhull

Back-end (lambdas):

- Custom OBJ file loader/dumper (package objio)
- Uses package markus-wa/quickhull-go for 3-D quickhull
- Uses AWS SDK (aws-sdk-go-v2) for lambda and s3 utilities
- Implements Lambda proxy integration

Components, cont'd.

Build system:

- Makefile with a lot of macros
- Uses aws-cli v2

Buckets:

- Allow public access via pre-signed GET and PUT requests
- Static (public) web hosting for front-end with index and error page

Things I wanted to try (but ran out of time to implement)

- Cognito user pools
  - OAuth2 Code Authorization Flow (w/ external identity providers)
  - Can allow authorization of API endpoints (like API key) using Access Token
  - Not too familiar with proper handling and verification of JWT tokens and Identity/Access/Refresh tokens
  - Requires HTTPS (Route 53)
- RDS
  - Store user and model information
- Route 53 and CloudFront
  - For DNS, HTTPS, and CDN services
  - Not on AWS Educate

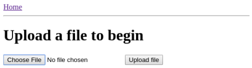
Build process

Top-level targets in the Makefile:

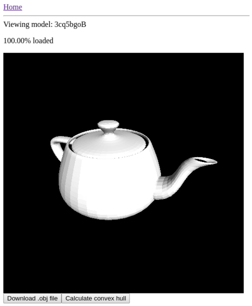
```
.PHONY:
all: build-website\
        host-bucket-create\
        host-bucket-sync\
        upload-bucket-create\
        upload-bucket-policy-create\
        lambda-iam-create\
        loggroup-create\
        lambda-create\
        api-create

.PHONY:
clean: target-clean\
        api-delete\
        lambda-delete\
        loggroup-delete\
        lambda-iam-delete\
        upload-bucket-policy-delete\
        upload-bucket-delete\
        host-bucket-delete
```

Demo

Demo link

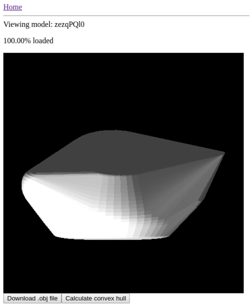

Home

Upload a file to begin


Home

Viewing model: 3cq5bgoB

100.00% loaded


Home

Viewing model: zezqPQl0

100.00% loaded

Lessons learned

To be more efficient:

- Use Amplify (Cognito/Route 53/Cloudfront integration)
- Use CDK (no shell/Makefile problems, full general-purpose language logic)

(but still a good learning experience w/o them)

Questions?

(Source code available on GitHub at @jlam55555/cloud-convex-hull.)