Geometrize Web Application Documentation

Comprehensive documentation covering use cases, benefits, technical architecture, workflow, code examples, and more.

1. Introduction

The Geometrize Web Application is a client-side tool that recreates images using geometric shapes. Based on the Geometrize Haxe project and the original Primitive algorithm, this web app allows users to upload or select an image, adjust parameters such as shape types and iterations, and generate a stylized version composed of simple geometric primitives. This documentation provides a comprehensive overview, including use cases, benefits, technical details, workflow, and examples.

2. Features and Options

- **Shape Types**: Circles, Triangles, Rectangles (rotated), Ellipses, Bezier Curves, and combinations.
- **Opacity (Alpha)**: Adjust the transparency of shapes (0-255).
- **Candidate Shapes per Step**: Number of random shape candidates evaluated each iteration.
- **Shape Mutations per Step**: Mutations applied to candidate shapes to refine fit.
- **Iterations**: Total number of shapes generated, impacting final detail and approximation quality.
- **Output Formats**: Export as SVG for scalable vector graphics, PNG for raster images, or JSON for custom rendering.
- **Live Preview**: Real-time visualization of shapes as they are added to the canvas.

3. Use Cases

1. **Digital Art Creation**: Artists can use the tool to create abstract renditions of photographs or designs.
2. **Educational Tool**: Demonstrating computational geometry, optimization, and image approximation in academic settings.
3. **Image Compression Preview**: Evaluate how few shapes capture the essence of an image, relevant to data visualization.
4. **Design Mockups**: Quickly generate stylized icons or backgrounds for web and graphic design.
5. **Algorithm Research**: Compare different optimization strategies by adjusting parameters and analyzing results.

4. Benefits

- **Visual Appeal**: Generates unique, abstract representations that can be more engaging than traditional filters.
- **Customization**: Fine-grained control over parameters allows tailored results for different needs.
- **Lightweight**: Client-side processing eliminates the need for backend servers, reducing infrastructure costs.
- **Open Source**: The underlying code is open, enabling extensibility and community contributions.
- **Cross-Platform**: Runs in modern web browsers without additional installations.

5. Technical Architecture

The Geometrize Web App consists of the following components:
- **Frontend (JavaScript/HTML/CSS)**: User interface built with standard web technologies.
- **Geometrize Haxe Library**: Core algorithm compiled from Haxe to JavaScript, responsible for generating shapes.
- **geometrizejs**: JavaScript bindings and utilities for interacting with the compiled library.
- **Rendering Engine**: Uses SVG elements or Canvas API to draw shapes in the browser.
- **Export Modules**: Convert the generated shape sequence into SVG, PNG (via Canvas), or JSON.
- **Optional Backend Services** (if used): While the core app is client-side, backends can be used for storing presets or sharing results.

6. Workflow

1. **Image Selection**: User uploads an image or selects one from available samples.
2. **Parameter Configuration**: Adjust shape types, candidate shapes per step, shape mutations, alpha, and iterations.
3. **Run Algorithm**: The Geometrize engine iteratively evaluates and adds shapes to minimize pixel difference.
4. **Live Preview**: As shapes are added, the canvas updates to show progress.
5. **Export**: Once complete (or at any intermediate step), user exports the result in desired format.
6. **Post-Processing**: Users can further edit exported SVG in vector editors or use JSON for custom rendering pipelines.

7. Code Examples

```
import Jimp from 'jimp'
import { Bitmap, ImageRunner, ShapeTypes, SvgExporter } from 'geometrizejs'

(async () => {
  const image = await Jimp.read('path/to/image.png')
    const  bitmap  =  Bitmap.createFromByteArray(image.bitmap.width,  image.bitmap.height,
image.bitmap.data)
  const runner = new ImageRunner(bitmap)
  const options = {
    shapeTypes: [ShapeTypes.CIRCLE, ShapeTypes.TRIANGLE],
    candidateShapesPerStep: 50,
    shapeMutationsPerStep: 100,
    alpha: 128
  }
  const iterations = 500
  const svgData = []
  for (let i = 0; i < iterations; i++) {
    svgData.push(SvgExporter.exportShapes(runner.step(options)))
  }
   const  svg  =  SvgExporter.getSvgPrelude()  +  SvgExporter.getSvgNodeOpen(bitmap.width,
bitmap.height)
    + svgData.join('\n') + SvgExporter.getSvgNodeClose()
  // Export logic here
})()
```

8. Screenshots

Below are examples of the application's interface at different stages. Replace these placeholders with actual screenshots when available:

- Screenshot 1: Initial parameter configuration UI.
- Screenshot 2: Live preview with shapes rendering in progress.
- Screenshot 3: Final exported SVG displayed in a vector editor.

9. Real-Life Applications

- **Marketing Materials**: Quickly generate stylized images for promotional content.
- **Game Development**: Create low-poly style textures or assets procedurally.
- **Data Visualization**: Represent complex data patterns through shape-based abstractions.
- **Educational Workshops**: Teach concepts of optimization, fitness evaluation, and computational graphics.

10. Conclusion

The Geometrize Web Application offers a versatile platform for exploring image approximation through geometric primitives. Its open-source nature, combined with comprehensive configuration options, makes it suitable for artists, educators, designers, and researchers. This documentation serves as a foundational guide to understand, utilize, and extend the application.

For further details or contributions, refer to the project's GitHub repository and accompanying API documentation.