

AAC(Ascii Art Converter)

2023 2학기, CSE2035/AIE2051
서강대학교



기말 프로젝트
Team: No More Double
<https://github.com/arduinooc04/AAC>

1st 조다니엘
컴퓨터공학과
서강대학교
danielcho@sogang.ac.kr

2nd 박준영
컴퓨터공학과
서강대학교
sparkjy18@sogang.ac.kr

Abstract—AAC(Ascii Art Converter)는 png 이미지 파일을 아스키 아트로 변환하여 출력하는 프로그램이다. 이미지 변환 방식은 크게 tone-based 방식과 structure-based 방식으로 나뉜다. tone-based 방식은 이미지의 각 픽셀의 rgba 값을 읽어들인 후 red, green, blue 값의 평균으로 픽셀의 밝기를 결정하여 색조 이미지를 회색조로 변환한다. 이후 픽셀의 밝기 정도에 따라 그에 해당하는 아스키 문자로 픽셀을 변환하여 출력한다. structure-based 방식은 이미지를 회색조로 변환한 뒤 convolution 연산을 활용한 edge-detection 알고리즘을 적용하며, morphology 또는 Gaussian-blur 연산을 적용하여 정제된 결과를 얻는다. 이후 Suzuki 알고리즘을 사용하여 외곽선을 추출한 후 외곽선의 곡선을 선분으로 분할하여 이미지를 벡터화한다. 최종적으로 벡터화된 이미지를 픽셀 둥어리로 분할한 다음 둥어리별로 log-polar 히스토그램을 얻고, Bhattacharyya 거리를 이용해 가장 유사한 아스키 문자를 선택하여 출력한다. 이후 simulated annealing 기법을 이용해 이미지가 아스키아트로 잘 표현되도록 이미지를 변형시키며, 더이상 변형되지 않을 경우 종료한다.

I. 동기

박준영: YouTube에서 접했던 donut.c [1] 아스키 아트를 보고 비슷하게 이미지 변환을 구현해 보고자 시작하였다.

조다니엘: AAC이라는 마음이 마음에 들었다.

II. 프로젝트 구조

A. eg

eg는 오픈 소스 라이브러리인 **libpng**와 **Eigen**을 사용하여 구현한 라이브러리이다. **eg**는 이미지 변환에 필요한 연산을 담당하는 **egGeometry**, **egMath**, **egMethods**, **egOperators**, **egProcessing**, **egTrace**와 그 밖의 이미지 입출력, 타입 정의, 여러 핸들링을 담당하는 **egLoader**, **egTypes**, **egExceptions**로 구분된다.

1) **egLoader**: **egLoader**에는 png 파일을 읽고, 저장하는데 도움을 주는 **libpng**를 이용하여 png 이미지 파일의 가로, 세로 및 rgba 값을 읽어들인 다음 각각 텐서의 원소로 저장한다. 또한 **Eigen**텐서 형태로 저장되어 있는 Image를 다시 png 파일로 export하는 기능이 내장되어 있다.

2) **egMethods**: **egMethods**는 지원 가능한 이미지 연산, 변환 방법을 enum 타입으로 정의한 파일이다. 대표적으로 색조 변환, blur, 윤곽선 추출 등이 있다.

3) **egMath**: **egMath**는 edge-detection에 활용되는 convolution 연산, 행렬 간의 거리를 비교하는 RMSE, shape-context, log-polar 좌표계 변환 연산, 히스토그램 간의 거리를 비교하는 Bhattacharyya 연산이 포함되어 있다.

4) **egGeometry**: **egGeometry**는 시계/반시계 방향을 판단하는 **ccw()** 함수 및 점과 선분 사이의 거리를 계산하는 **distSegDot()** 함수, 내적 및 외적, 유clidean 좌표계의 거리와 log-polar 좌표계의 거리를 계산하는 함수들이 내장되어 있다.

5) **egProcessing**: **egProcessing**은 이미지 처리와 관련된 전반적인 구현이 모두 포함되어 있다. 색조 이미지를 회색조로 변환하는 **cvtGray()**, 외곽선을 추출하는 **getEdge()**, blur 처리를 담당하는 **blur()**, grassfire 알고리즘을 활용하여 중심선을 추출하는 **extractCenterline()**, 0과 255 사이를 벗어나는 밝기 값을 범위 안으로 축소하는 **saturate()**, 일정 역치를 벗어나는 이미지의 명도 픽셀 값을 표시하는 **markOutlier()**, binary 이미지의 0과 1값을 반전시키는 **reverse()**, 이미지 morphology 연산에 활용하기 위하여 구현한 **erode()** 및 **dilate()**, Suzuki 알고리즘을 이용하여 외곽선을 추출하는 **getContours()**, 개발하는 대로 계속 수정해 주세요

6) **egTrace**: **egTrace**는 **distSegDot()** 등 곡선을 선분으로 분할하는 함수를 사용하여 이미지의 윤곽선을 여러 선분으로 잘게 잘라 벡터화한다.

- 7) **egTypes**: **egTypes**에는 프로젝트에서 사용하기 위해 새롭게 정의한 타입들의 정보를 한데 모아 놓았다.
- 8) **egOperators**: **egOperators**에는 픽셀의 x좌표, y좌표를 각각 더하고 빼거나 부호를 반전할 수 있도록 연산자 오버로딩을 정의해 놓았다.
- 9) **egExceptions**: **egExceptions**는 프로그램을 실행 중에 오류가 발생하면 에러 코드를 throw하고 작동을 멈춤으로써 프로그램의 undefined behavior를 방지하는 역할을 한다.

B. img2ascii

img2ascii는 **eg**라이브러리를 바탕으로 이미지를 아스키 아트로 변환하는 과정을 기술한 소스코드이다. tone-based 방식으로 변환하는 **tone**과 structure-based 방식으로 변환하는 **structure**로 나뉜다.

1) **tone**: tone-based 방식 변환은 회색조 변환 단계 이후 각 픽셀별 red, blue, green 값의 평균을 이용하여 밝기를 추출하고, 해당 밝기의 정도에 따라 문자를 선택하여 출력한다.

2) **structure**:

III. 프로그램의 동작

이번 절에서는 structure-based 방식의 이미지 변환을 설명한다.

A. 이미지 파일의 입력

이미지 파일을 입력받는다. 이미지 파일의 형식이 png인지 확인하고, 형식에 맞지 않는 이미지 파일은 오류를 띠운다. 이는 **egLoader**에 구현된 PNG 클래스에서 일어나는 일이며, 특히 `openImage()` 메서드...

B. 이미지 전처리

1) 회색조 변환: 색조 이미지의 r, g, b값을 평균내서 회색조 이미지로 변환한다.

2) 윤곽선 추출: 회색조로 변환된 이미지의 윤곽선을 추출한다.

3) Blurring or Morphology: Gaussian-blur 또는 dilate, erode 연산을 이용해 노이즈를 줄인다.

4) binary 변환: 이미지를 공백에 해당하는 0과 선에 해당하는 1로 표현하여 2차원 행렬에 저장한다. 일정 역치 이상

5) 외곽선 추출: Suzuki의 알고리즘 [2]을 이용하여 외곽선을 추출한다.

6) 점의 집합을 선분으로: suzuki의 알고리즘을 이용하면 이차원 행렬의 각 점이 어떤 외곽선에 속해있는지를 알아낼 수 있다. 이를 바탕으로 실제 외곽선을 구해내려면, 같은 외곽선에 있는 점을 시계방향 또는 반시계방향으로 정렬해야 한다. 이를 위해, 알고리즘을 자체 제작하였다.

- 1) 원쪽 위(0, 0) 부터 오른쪽 아래(H, W) 방향으로 행렬의 각 원소를 검사한다.
- 2) 만약 그 픽셀의 값이 이전에 봤던 값이거나 0 이라면 1로 돌아간다.
- 3) 자신과 인접한 원쪽 아래에 있는 픽셀부터 시계 반대 방향으로 자신과 같은 외곽선에 속한 픽셀을 찾는다. 없다면 1로 돌아간다.
- 4) 이후 연결된 픽셀을 순차적으로 탐색하여, 연결된 픽셀이 더이상 없을때까지 스택에 픽셀의 좌표를 넣는다.
- 5) 스택을 뒤집는다.

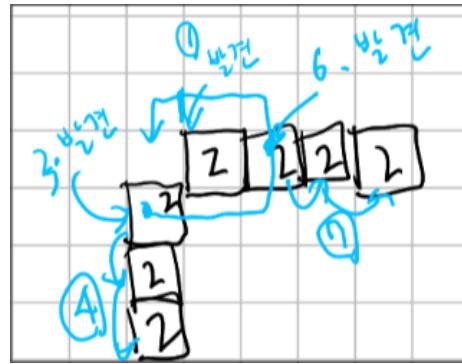


Fig. 1. 점의 집합을 시계방향 또는 시계 반대방향으로 정렬하는 알고리즘

- 6) 3에서 발견했던 픽셀의 시계 반대방향 순서로 다음에 있는 픽셀부터, 시계 반대방향 순서로 탐색을 시작한다. 만약 기준에 방문한 픽셀 밖에 없거나, 같은 외곽선에 속한 픽셀을 찾지 못한다면 1로 돌아간다.
- 7) 이후 연결된 픽셀을 순차적으로 탐색하여, 연결된 픽셀이 더이상 없을때까지 스택에 픽셀의 좌표를 넣는다. 이후 1로 돌아간다.

이후, 간단한 그리디 알고리즘을 이용하여 점들을 선분으로 균사한다. 선분 $\overline{A_1 A_2}$ 와 점 P 사이의 거리는 식 1로 정의한다.

$$\begin{cases} \frac{A_1 P}{A_2 P} & \text{If } (A_2 - A_1) \cdot (P - A_1) \leq 0 \\ \frac{A_2 P}{A_1 P} & \text{If } (A_1 - A_2) \cdot (P - A_2) \leq 0 \\ \frac{|(P - A_1) \times (A_2 - A_1)|}{A_1 A_2} & \text{otherwise} \end{cases} \quad (1)$$

시계 방향 또는 시계 반대 방향으로 정렬된 점들의 배열 A 가 주어졌을 때, 어떤 $i < j$ 인 음이 아닌 정수 i, j 가 존재하여, 선분 $\overline{A_i A_j}$ 와 점 A_k ($i < k < j$)의 거리가 $\sqrt{2} \approx 1.42$ 이하라면, i 에서 j 번째까지의 점들을 선분으로 균사했다고 말할 수 있을 것이다. 이를 배열 앞쪽부터 보면서 선분으로 모을 수 있을 때는 항상 모으고, 아니면 빠르게 포기하고 새로운 선분을 만들기 시작한다.

C. 이미지 최적화

1) 선분을 그리드로 분할: Cohen-Sutherland 알고리즘 [5]을 사용하여 기다란 선분을 그리드마다 분할해둔다.

2) simulated annealing: 임의의 정점을 조금씩 움직여보면서 Bresenham's line algorithm [7]을 사용해 그림을 새로 그린다. 이후 아스키아트를 더 잘 표현하는 이미지로 변화했는지 확인한다. 주어진 그리드에 어떤 문자가 가장 잘 대응되는지는 log polar coordinate로 변환한 후 히스토그램을 만든 후, Bhattacharyya 거리를 비교하여 판단한다. 자세한 내용은 [6]로..

IV. 프로젝트 진행 과정

프로젝트의 첫 계획은 이미지를 아스키 아트로 변환하는 프로그램과 변환된 아스키 아트를 출력하는 프로그램 두 가지를 결합하는 형식이었다. 그러나 Qt로 뷰어를 구현하는 과정은 번잡하고 지루하였기 때문에 아스키 아트 변환에만 집중해서 프로젝트를 진행하기로 결정하였다. 방식을 일반적으로 널리 알려지고 구현 난도도 낮은 tone-based 방식과 구현 난이도가 높지만 훨씬 만족스러운 결과를 도출하는 structure-based 방식으로 나누어 각자 구현하기로 하였다. 처음에는

libpng를 제외하고 아무런 오픈 소스 라이브러리를 사용하지 않으려고 하였기 때문에 이미지를 쉽고 빠르게 입출력할 수 있는 사용자 라이브러리를 만들었다.

V. 사용 방법

- 1) sudo pacman -S libpng eigen
- 2) git clone https://github.com/arduinocc04/AAC
- 3) cd AAC
- 4) ./install.sh
- 5) cd build
- 6) img2ascii/structure/sa-structure non-hangul-images IMAGE_FILE

REFERENCES

- [1]
- [2]
- [3]
- [4]
- [5]
- [6]
- [7]
- [8]