

AAC - Ascii Art Converter

No More Double

조다니엘, 박준영

Sogang University

CSE2035/AIE2051

2023년 12월 9일



목차

- ① 왜 만들었는가?
- ② 무엇으로 만들었는가?
 - eg
 - img2ascii
 - 기타
- ③ 어떻게 동작하는가?
 - tone-based 방식
 - structure-based 방식

왜 만들었는가?

- AAC(악)이라는 이름이 너무 찰져서
- YouTube에서 donut.c라는 아스키 아트를 봤는데 멋있어서¹



¹https://www.youtube.com/watch?v=DEqXNfs_HhY

무엇으로 만들었는가?

- ① 자체 제작 라이브러리: eg
- ② 메인 루틴: **img2ascii**
- ③ 기타: GitHub, Valgrind, ...

eg

Easy pn**G**

libpng + Eigen으로 구성한 자체 제작 라이브러리
이미지 입출력, 변환, 연산 등에 필요한 클래스를 내장

- egLoader: 이미지 입출력
- egMath: 변환에 필요한 연산 내장
(convolution, Bhattacharyya 거리)
- egProcessing: 이미지 변환
(회색조, 흑백, blur, Suzuki, erode, dilate, grassfire)
- egGeometry: 외적, 내적, norm 및 거리 계산
- egTrace: 이미지 벡터화

img2ascii

eg 라이브러리를 활용하여 이미지를 처리하는 메인 루틴

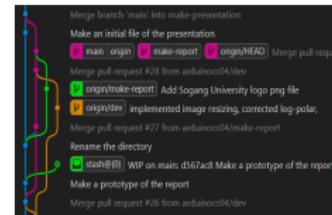
tone-based 방식과 **structure-based** 방식으로 나뉨

(**tone-based** 방식 결과물 사진 한 장, **structure-based** 방식
결과물 사진 한 장 넣기)

기타

- GitHub: 소스 코드 버전 관리 플랫폼

GitHub



- Valgrind: 메모리 누수 탐지 라이브러리

```
=6042= HEAP SUMMARY:
=6042=     in use at exit: 14,621,750 bytes in 868 blocks
=6042=   total heap usage: 2,240,428 allocs, 2,239,560 frees, 606,475,983 bytes allocated
=6042=
=6042= 36,456 bytes in 93 blocks are definitely lost in loss record 29 of 38
=6042=    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
=6042=    by 0x4868C09: png_create_info_struct (in /usr/lib/x86_64-linux-gnu/libpng16.so.16.37.0)
=6042=    by 0x1147C4: eg::PNG::readImageBuffer(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>)
=6042=    by 0x114B2F: eg::PNG::openImage(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>)
=6042=    by 0x10D2A6: getAllImages(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>)
```

어떻게 동작하는가?

- tone-based 방식 - easy
- structure-based 방식 - DIFFICULT

tone-based 방식

- ① 이미지를 3차원 텐서 ($width, height, rgba$) 형태로 저장
- ② 각 픽셀별 red, green, blue 값의 평균을 구하여 밝기 도출
- ③ 밝기 정보를 이용하여 이미지를 회색조로 변환
- ④ 밝기에 해당하는 아스키 문자를 출력

결과



→

The image is a high-contrast, black-and-white graphic. It features a dense, organic cluster of geometric shapes, predominantly triangles and circles, arranged in a non-linear, branching pattern. The shapes vary in size and orientation, creating a sense of depth and complexity. The background is a solid, light color, which makes the dark shapes stand out. There are no discernible text elements or other specific symbols.

structure-based 방식

- ① 이미지 저장
- ② 회색조 변환
- ③ Binary 변환
- ④ 외곽선의 선분 근사
- ⑤ 아스키 문자로 최적화

이미지 저장

이미지를 3차원 텐서 ($height, width, rgba$) 형태로 저장

```
using Image = Eigen::Tensor<png_byte, 3>;
```

```
image.dimensions()[0]: height
```

```
image.dimensions()[1]: width
```

```
image(i, j, 0): red
```

```
image(i, j, 1): green
```

```
image(i, j, 2): blue
```

```
image(i, j, 3): alpha
```

회색조 변환

- 색조 이미지를 회색조로 변환
- red, green, blue 값의 평균을 밝기로 치환
- tone-based 방식과 동일

$$\frac{red + green + blue}{3} = brightness$$

Binary 변환

① Gaussian-blur → Binary

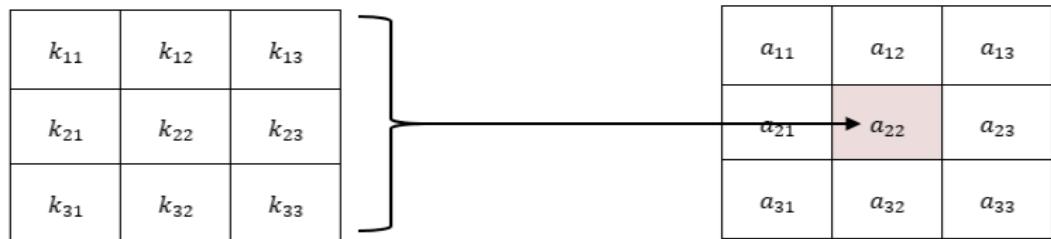
- 3×3 Gaussian 필터를 이용해 convolution 연산을 적용
- 이미지를 흐리게 처리

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

Gaussian 필터

Binary 변환

- Convolution 연산

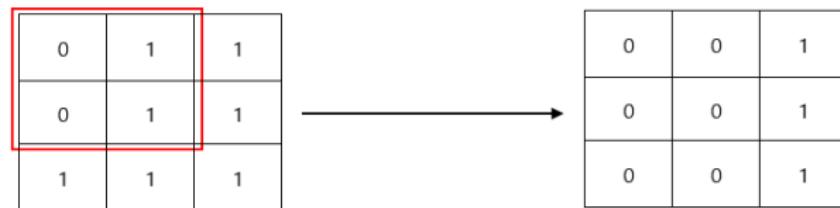


$$a_{22} = \sum_{i=1}^3 \sum_{j=1}^3 a_{ij} * k_{ij}$$

Binary 변환

② Binary → Morphology

- erosion: 범위에 0이 존재하면 0으로 변환



- dilation: 범위에 1이 존재하면 1로 변환



외곽선의 선분 근사

- ① Suzuki 알고리즘을 이용하여 외곽선을 추출
- ② 자체 제작한 CW 정렬 알고리즘을 이용하여 외곽선을 선분으로 분할

외곽선의 선분 근사

CW 정렬 알고리즘

- ① 왼쪽 위 $(0, 0)$ 부터 오른쪽 아래 (H, W) 방향으로 행렬의 각 원소를 검사한다.
- ② 현재 픽셀의 왼쪽 아래에 위치한 픽셀부터 시계 반대 방향으로 자신과 같은 외곽선에 속한 픽셀을 찾는다.
- ③ 연결된 픽셀을 순차적으로 탐색하며 연결된 픽셀이 더 이상 없을 때까지 스택에 픽셀의 좌표를 넣는다.

외곽선의 선분 근사

- ④ 스택을 뒤집는다. ($CCW \rightarrow CW$)
- ⑤ 2에서 얻은 픽셀의 시계 반대 방향에 위치한 다음 픽셀부터 시계 반대 방향으로 탐색을 시작한다. 만약 기존에 방문한 픽셀밖에 없거나, 같은 외곽선에 속한 픽셀을 찾을 수 없다면 1로 되돌아간다.

아스키 문자로 최적화

- ① Cohen-Sutherland 알고리즘을 사용하여 길이가 긴 선분을 일정한 간격의 그리드로 분할
- ② Simulated annealing(담금질 기법): 임의의 정점을 조금씩 움직이면서 Bresenham's line algorithm을 사용해 그림을 새로 그림
- ③ 새로 그린 이미지가 아스키 아트로 더 잘 표현되는 결과물로 변화하였는지 확인

아스키 문자로 최적화

- ④ 주어진 그리드에 어떤 문자가 가장 잘 대응되는지 판단하기 위하여 좌표계를 log-polar coordinate로 변환한 후 히스토그램을 만듦
- ⑤ 히스토그램 간의 Bhattacharyya 거리를 비교하여 최적의 아스키 문자를 판단

Bhattacharyya 거리:

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H}_1 \bar{H}_2 N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}}$$

- 히스토그램을 전체 크기로 나누어 합이 1인 확률분포로 변형
- 확률분포화된 히스토그램의 유사도를 계산

결과

(처음 input 사진 한 장 → 아스키 아트 사진 한 장)

참고 자료

① 자료 1

② 자료 2

③ 자료 3

(인용 양식 지켜서 쓰면 교수님 호감 → 대학원생으로 바로 납치)

감사합니다.