Universitatea Tehnică "Gheorghe Asachi" din Iași
Facultatea de Automatică și Calculatoare
Prelucrarea Imaginilor - Proiect

Raport intermediar - Automatic 2D-to-3D image conversion

ECHIPĂ: E6
Beldiman Vladislav Student1
Grupa 1305A

27 noiembrie 2020

1 Descrierea temei

Obiectivul acestui proiect este de a crea o aplicație Windows cu interfață grafică care realizează conversia unei imagini din 2D în 3D cu interacțiune minimă în cadrul acestui proces din partea utilizatorului. Acest lucru va fi realizat considerând intensitatea fiecărui pixel drept valoarea înălțimii acestuia într-un câmp de înălțimi, iar pe baza acestor valori va fi construită o plasă poligonală cu fețe triunghiulare (Figura 1).

Aplicația va fi scrisă în limbajul de programare C++ cu interfața grafică realizată cu ajutorul setului de instrumente Qt, iar plasa poligonală va fi sintetizată folosind interfața pentru programarea aplicațiilor OpenGL. Ea va permite vizualizarea imaginii încărcate și finale, cât și salvarea imaginii 3D în format STL.

Utilizatorul va avea la dispoziție următoarele opțiuni pentru calibrarea rezultatului final:

- Selectarea algoritmului de triangulație;
- Selectarea erorii maxime la triangulație (unde e cazul);
- Inversarea imaginii finale (răsturnarea valorilor din câmp);
- Adăugarea unui chenar cu grosime și înălțime configurabile la imaginea finală;
- Adăugarea unei baze la imaginea finală cu înălțime configurabilă;
- Selectarea înălțimii dintre valoarea maximă și minimă de gri.

Triangulația domeniului determinat de pixeli va fi realizată prin triangulația naivă (Figura 2) care include toate punctele corespunzătoare pixelilor, sau prin aproximare prin înserare lacomă (Figura 3) aplicând triangulația Delaunay sau triangulația dependentă de date cu unele optimizări [1]. Înserarea lacomă va folosi drept măsură de importanță pentru fiecare punct eroarea verticală dintre valoarea câmpului și aproximarea interpolată în acel punct.

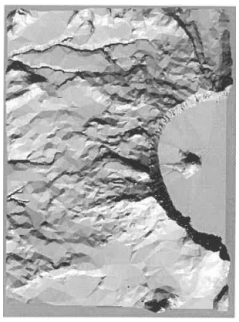

Figura 1: Exemplu de plasă poligonală rezultantă. [1]

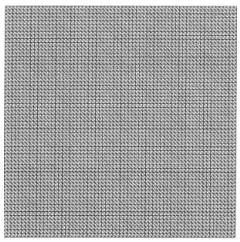

Figura 2: Exemplu de triangulație naivă. [1]

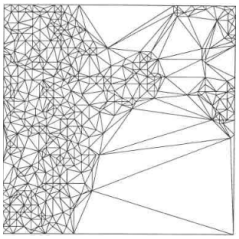

Figura 3: Exemplu de triangulație care aproximează un câmp de înălțimi. [1]

2 Modalitatea de lucru propusă

Git repository: https://github.com/veeyslaw/hfbm

#	Descriere sarcină	Stare	Membru
1	Documentare despre conversia imaginilor din 2D în 3D	în progres	m1
2	Documentare despre Qt	în progres	m1
3	Documentare despre OpenGL	în progres	m1
4	Implementarea și testarea unei interfețe grafice minimaliste	încheiată	m1
5	Implementarea și testarea algoritmului naiv	încheiată	m1
6	Documentare despre formatul StL	încheiată	m1
7	Implementarea și testarea opțiunii de salvare în format StL a imaginii 3D	încheiată	m1
8	Întocmirea raportului intermediar	încheiată	m1
9	Extinderea interfeței grafice cu opțiuni de calibrare și perfectarea acesteia	în progres	m1
10	Implementarea și testarea opțiunilor de calibrare	-	m1
11	Implementarea și testarea algoritmului bazat pe triangulația Delaunay	-	m1
12	Implementarea și testarea algoritmului bazat pe triangulația dependentă de date	-	m1
13	Rafinare și optimizări	în progres	m1
14	Identificarea unui set potrivit de imagini (preferabil găsirea unei surse cu rezultate experimentale împreună cu probele folosite)	-	m1
15	Adăugarea posibilității de cronometrare a timpului de conversie	-	m1
16	Realizarea experimentelor	-	m1
17	Întocmirea raportului final	-	m1
18	Pregătirea prezentării	-	m1

3 Rezultate și concluzii

3.1 Implementarea și testarea unei interfețe grafice minimaliste

Interfața grafică a fost implementată cu ajutorul setului de intrumente Qt. În timpul implementării am realizat faptul că QMesh - clasa pentru un încărcător de plase personalizate pe care intenționam să o folosesc - nu oferă la fel de multe facilități la câte mă așteptam. Astfel, am decis să folosesc OpenGL pentru sinteti-

zarea plasei prin interfața oferită de Qt într-un QOpenGLWidget.

La acest pas interfața permitea încărcarea și vizualizarea imaginii de intrare încărcate într-un QLabel și vizualizarea plasei rezultante care permite și scalarea ei cu ajutorul roții șoricelului, cât și rotirea aceasteia pe axa Ox și Oy. Însă, vizualizarea plasei mai are unele probleme care sunt cauzate de iluminare necorespunzătoare și care vor fi rezolvate la sarcina de perfectare a interfeței grafice. De asemenea, au fost conectate butoanele adăugate la funcțiile corespunzătoare care urmau să fie implementate.

Rezultate: Figura 4

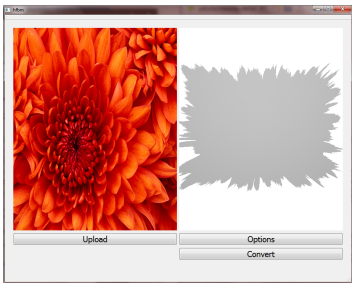

Figura 4: Pagina principanlă după prima etapă.

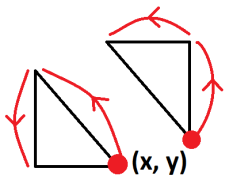

Figura 5: Alegerea triunghiurilor triangulația naivă.

3.2 Implementarea și testarea algoritmului naiv

Algoritmul naiv a fost implementat și testat cu succes. În cadrul algoritmului este adăugat câte un punct pentru fiecare punct din imaginea sursă având ca valoare pe axa Oz valoarea de gri a imaginii normalizată. După care, pentru fiecare punct în afară de cele de pe prima coloană și primul rând sunt adăugate câte 2 triunghiuri în tabloul de indici după cum e prezentat în figura 5.

Rezultate: Figurile 6 7 8 (Rezultatele plasei în afară de primul sunt prezentate într-un vizualizator StL online până când vor fi rezolvate problemele cu iluminația.)

3.3 Implementarea și testarea opțiunii de salvare în format StL a imaginii 3D

În timpul rezolvării acestei sarcini, cât și a documentării despre OpenGL, am stabilit cum vor fi stocate datele plasei în timpul triangulației și după, și anume:

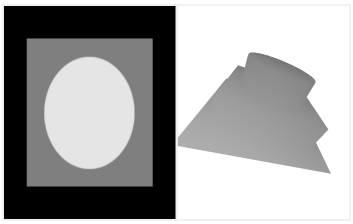

Figura 6: Exemplu triangulație naivă 1.

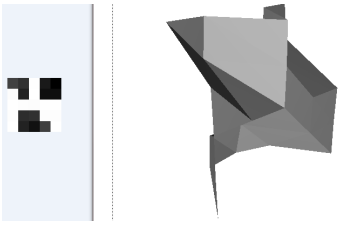

Figura 7: Exemplu triangulație naivă 2.

Figura 8: Exemplu triangulație naivă 3.

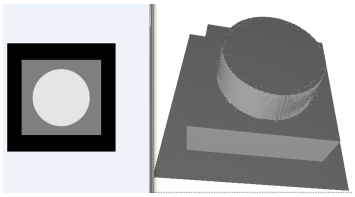

Figura 9: Exemplu salvare și afișare în altă aplicație.

coordonatele punctelor într-un vector de tridimensional, la care va fi adăugată după finisarea algoritmului date despre culorile și normalele cu scopul de a le sintetiza în QOpenGLWidget, iar în loc de a stoca majoritatea punctelor de 2 ori, va fi folosit un tablou de indici, câte 3 pentru fiecare triunghi, fiecare din ei reprezentând poziția în tabloul de puncte a punctelor care îl alcătuiesc, eliminând nevoia de a stoca câte 2 puncte pentru fiecare latură comună din plasă. În plus, OpenGL ne permite să folosim și această reprezentare la tamponarea datelor pentru sintetizare. Indicii vor apărea în ordine trigonometrică, ordine conformă cu standardul StL [2], și cu modul implicit în care detectează OpenGL orientarea triunghiurilor.

Conform standardului StL [2]:

- Un fișier StL constă dintr-o listă de date despre fațete;
- Orientarea fațetelor e specificată redundant în două moduri - prin direcția normalei și prin ordinea în care sunt specificate vârfurile - trigonometrică;
- Fiecare triunghi trebuie să împartă două vârfuri cu fiecare două triunghiuri vecine;
- Formatul binar e compus dintr-un antet pe 80 de bytes care nu are nici o semnificație de dată, 4 bytes de tipul unsigned long integer - numărul de fațete în fișier, după care pentru fiecare triunghi sunt adăugate datele: 3 x 4

bytes de tip float care reprezintă vectorul normalei, câte 3 x 4 bytes de tip float pentru toți cei 3 vectori - vârfuri, și 2 bytes de tip unsigned integer care nu vor avea vreo însemnătate în cadrul proiectului;

Rezultate: Figura 9 (Rezultatul e prezentat într-un vizualizator StL online, deoarece încărcarea unui fișier StL și vizualizarea acestuia nu se include în scopul proiectului.)

3.4 Extinderea interfeței grafice cu opțiuni de calibrare și perfectarea acesteia

A fost adăugată cea mai mare parte a interfeței grafice pentru selectarea opțiunilor de calibrare (Figura 10).

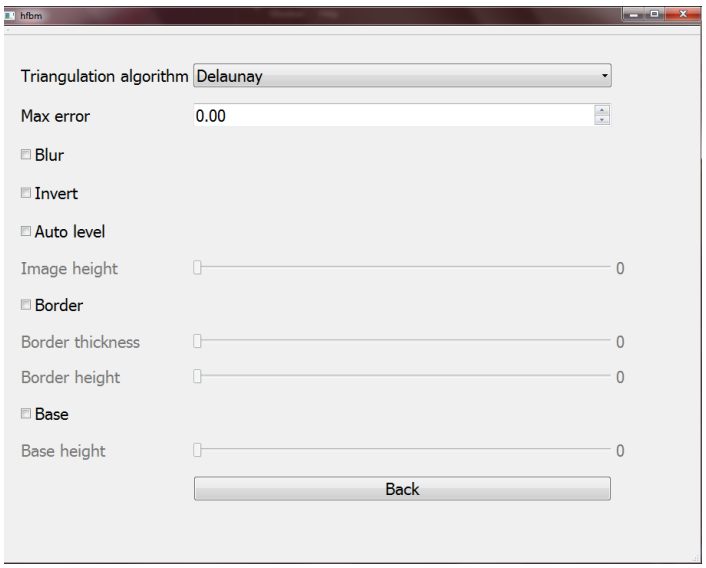

Figura 10: Pagina cu opțiuni curentă.

3.5 Implementarea și testarea opțiunilor de calibrare

-

3.6 Implementarea și testarea algoritmului bazat pe triangulația Delaunay

-

3.7 Implementarea și testarea algoritmului bazat pe triangulația dependentă de date

-

3.8 Identificarea unui set potrivit de imagini (preferabil găsirea unei surse cu rezultate experimentale împreună cu probele folosite)

-

Referințe

[1] Garland M. and Heckbert P. S. (1995) Fast Polygonal Approximation of Terrains and Height Fields. (CMU-CS-95-181)

[2] Marshall Burns. Automated Fabrication Improving Productivity in Manufacturing.