УЧЕНИЧЕСКИ ИНСТИТУТ ПО МАТЕМАТИКА И ИНФОРМАТИКА

ПЕТНАДЕСЕТА УЧЕНИЧЕСКА СЕКЦИЯ УС'15

ТЕМА НА ПРОЕКТА
Astro Spiral

Автор:
Галина Цанкова Станева
МГ "Д-р Петър Берон", Варна, 8 клас

Научен ръководител:
доц. д-р Галина Момчева
ВСУ "Черноризец Храбър"

Научен консултант:
д-р Веселка Радева
НАОП "Николай Коперник", Варна

Абстракт

Проектът *Astro Spiral* представя иновативен подход при сравняването на астрономически снимки на небето чрез построяване на изпъкнала спирала върху светлите обекти от такава снимка. За тази цел, написахме програма на *C++* с библиотеката *Cinder*. Тя намира светлите обекти в астрономическа снимка в JPEG формат и ги свързва в /emphизпъкнала обвивка чрез собствен алгоритъм на базата на *Graham Scan* за изпъкнала обвивка. Планираме създаването на сайт, в който астрономи да използват разработените алгоритми.

Абстракт

The *Astro Spiral* project presents an innovative way to compare astronomical photos of the sky by building a *convex spiral* according to the bright objects in a photo. On that purpose, we made an application for *Windows* and *Mac OS X*, written in *C++* that uses the *Cinder* library. We are planning to make a site where astronomers could use our algorithms.

Съдържание

1 Въведение **5**

2 Особености **5**
2.1 Снимки от телескоп 5
2.1.1 CCD камера на телескоп 5
2.1.2 Цвят 6
2.1.3 Размер 6
2.1.4 Последователно взети астрономически снимки на небето 6
2.1.5 Примерни снимки от телескоп на НАО – Рожен 6
2.1.6 Формат *FITS* 6
2.2 C++ с Cinder 6

3 Алгоритъм за изпъкнала обвивка *Греъм Скан* **8**
3.1 Изпъкнала обвивка 8
3.2 Алгоритъм 8
3.2.1 Стъпки 8
3.2.2 Реализация на *C++* 9
3.3 Сложност на алгоритъма 11

4 Алгоритъм за постряване на изпъкнала спирала (авторски) **11**
4.1 Стъпки 11
4.1.1 Реализация на *C++* в програмата ни 13

5 Превръщане на астрономическа снимка в монохромно изображение (авторски алгоритъм) **14**
5.1 Стъпки 14
5.2 Реализация на *C++* в програмата ни 15
5.3 Пример 15

6 Определяне на един пиксел от всеки бял фрагмент (авторски алгоритъм) **15**
6.1 Стъпки 16
6.2 Реализация на *C++* в програмата ни 17

7 Алгоритъмът на Брезенхам за строене на отсечка между два пиксела **19**
7.1 Реализация на *C++* в програмата ни 19

8 Работа с пректа **21**

9 Примери **22**

10 Потребители **24**

11 Проблеми **26**

12 Заключение **27**

Списък на фигурите

2 Изпъкнала обвивка на мнжество от точки 8
5 Отваряме проекта 22
6 Кодът на проекта 23
7 Кодът на проекта 23
8 Отваряне на снимка 24
9 Резултатно изображение 24
10 Запазеното изображение 25

1 Въведение

Астроинформатиката е една нова научна област, която обединява астрономия и информатика. Непрестанното развитие на астрономията изисква работа с все по-голямо количество данни. *Астроинформатиката* дава възможност за създаване на нови методи и софтуер, които да помагат на астрономите да анализират и обработват астрономически данни.

Астроинформатиката намира много приложения в астрономията, едно от които е изследването на небето чрез обработване на астрономически снимки. По този начин се откриват обекти, като например нови астероиди.

Астероидите са малки странстващи обекти в космоса, които обикалят около слънцето. Думата *астероид* означава *подобно на звезда*. Това е така, защото астероидите светят като звезди, и, гледайки небето, е трудно да ги различим.

Учените изследват небето в търсене на астероиди от близо 200 години, но още не са успели да намерят всички астероиди, обикалящи около слънцето. Търсенето на астероиди от Земята е много удобно и важно за развитието на астрономията и други науки. В последните години тази тема става много популярна, защото технологиите предоставят невиждани досега възможности за търсене и намиране на астероиди.

Задачата, която си поставяме в тази разработка, е да създадем софтуер, който строи изпъкнала спираловидна фигура по звездоподобните обекти на астрономическа снимка на небето. При поредица от астрономически снимки, взети последователно от един и същи телескоп, от едно и също място, ние можем много интуитивно, чрез гледане, да намерим разликите в спиралите и по този начин да открием движещи се, подобни на звезди обекти, които са най-често астероиди.

Нашият метод за строене на изпъкнали спирали представя иновативен подход за сравняване на астрономически снимки, който е много полезен за намирането на астероиди. Програмата ни приема астрономически снимки в JPEG формат, което ги прави по-достъпни за любители-астрономи, които не притежават професионално оборудване.

2 Особености

Тук разглеждаме някои особености от астрономията и програмите ни, които са от изключително важни за разбирането на проекта ни.

2.1 Снимки от телескоп

2.1.1 CCD камера на телескоп

Астрономическа CCD (Charge Couple Device) камера на телескоп е електронен сензор за цифрова фотография, използван за документиране на научни данни. *CCD камерите* използват *CCD матрица* - силициеви двумерни ма-

сиви, които съхраняват информация за изображение, като преобразуват падащите по повърхността им фотони в индуцирани електрични заряди.

2.1.2 Цвят

Снимките, направени от професионален телескоп с *CCD камерата* му са черно-бели. По-късно можем до им добавим цвят със специални програми, като например *MaxIm DL*.

2.1.3 Размер

Размерът на снимките зависи от *CCD камерата* на телескопа, с който са снимани.

2.1.4 Последователно взети астрономически снимки на небето

Когато снимаме небето с телескоп, можем да определяме през какъв интервал от време ще вземаме снимка (например през една, три, пет или десет минути). На тези последователно взети астрономически снимки виждаме местенето на звездите в небето спрямо нас и намираме астероиди чрез анализирането на необичайни движения.

2.1.5 Примерни снимки от телескоп на НАО – Рожен

Снимки от телескоп на *(Национална Астрономическа Обсерватория –(НАО) — Рожен* виждаме на фигура 1а, 1б и 1в.

2.1.6 Формат *FITS*

Снимките, направени от професионален телескоп, са във формат *Flexible Image Transport System (FITS)* Този формат е възприет от *Международния астрономически съюз (IAU)*. Нашите програми приемат изображение в формат *JPEG (Joint Photographic Experts Group)*. За да конвертираме изображение от *FITS* в *JPEG* формат, можем да използваме инструменти от страницата на *NASA*[**?**].

2.2 C++ с Cinder

C++ е много подходящ език за програмата ни. Базиран на *C*, той позволява ниско ниво на достъп до команди, докато все още запазва преносимостта и синтаксиса си. Езикът осигурява бързо изпълнение на програмите.

C++ поддържа множество от *библиотеки*, *типове*, *функции* и *операции*, които са много полезни за нашия алгоритъм. Затова избираме *C++* за написването на нашия *алгоритъм за спирала*.

Cinder е библиотека към *C++*, която представя множество функции за графика, изчислителна геометрия и обработка на изображения. Приложенията, направени с Cinder, работят за Windows, Mac OS X, iPhone и iPad, което ги прави независими.

(а) Първа снимка от телескоп на НАО – Рожен

(б) Втора снимка от телескоп на НАО – Рожен

(в) Трета снимка от телескоп на НАО – Рожен

Затова избираме C++ с библиотеката Cinder за направата на нашето приложение.

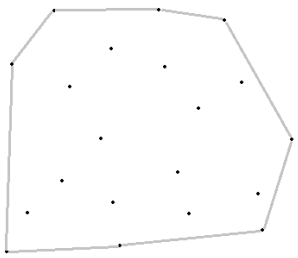
3 Алгоритъм за изпъкнала обвивка *Греъм Скан*

3.1 Изпъкнала обвивка

Изпъкнала обвивка за множество от точки в равнината е изпъкнал многоъгълник с минимално лице, който съдържа в себе си или като свои върхове цялото множество от точки.

Пример за изпъкнала обвивка можем да видим на фигура 2.

Намираме много алгоритми за изпъкнала обвивка, които не се различават особено по идея и сложност. Избрахме алгоритъма *Graham Scan* освен заради бързината, заради възможността за модифициране на този алгоритъм до алгоритъм за намиране на *изпъкнала спирала* за дадено множество от точки в равнината. Началната част на алгоритъма съдържа сортиране на точките спрямо една определена, което се повтаря многократно в авторския алгоритъм за строене на *изпъкнала спирала*.



Фигура 2: Изпъкнала обвивка на мнжество от точки

3.2 Алгоритъм

Тук представяме алгоритъмът *Graham Scan* за строене на изпъкнала обвивка на множество от точки в равнината.

3.2.1 Стъпки

Началото на координатната система, която използваме, се намира в центъра на равнината.

- Намираме най-долната точка, т.е. тази с минимална *y координата*. Ако има повече от една такива точки, избираме най-лявата от тях, т.е. тази с минимална *x координата*.

- Разменяме първата точка с намерената.
- Сортираме останалите точки по ъглите на всяка спрямо първата.
- Добавяме за последна точка такава като първата.
- Запчваме обхождане на точките от третата до последната точка.

  Всеки път проверяваме дали предните две намерени точки от обвивката и текущата лежат на една права или детерминантата им е положителна.

  Когато намерим такава точка, разменяме следващата с текущата в последовтаелността.

  При всяка размяна увеличаваме броячът на елементи от изпъкналата обвивка k, който в началото има стойност 2 (започваме от третия елемент).
- Накрая извеждаме първите k елемента от последователността.

3.2.2 Реализация на *C++*

```
#include<iostream>
#include<vector>
#include<algorithm>
#include<cmath>
using namespace std;

typedef struct
{
    double x, y;
    int num, orient;
} point;

vector<point> p;
point pp;
int n;

bool cmp(point a, point b)
{ return atan2(a.y-p[0].y, a.x-p[0].x)<=atan2(b.y-p[0].y, b.x-p[0].x); }

int det(point a, point b, point c)
{
    int dxa=a.x-c.x, dya=a.y-c.y;
    int dxb=b.x-c.x, dyb=b.y-c.y;
    return (dxa*dyb - dxb*dya);
}
```

```
void view()
{
    for(int i=0; i<n; i++)
        cout<<p[i].num<<" ";
    cout<<endl;
}

int main()
{

    cin>>n;
    for(int i=0; i<n; i++)
    {
        cin>>pp.x>>pp.y;
        pp.num=i+1;
        p.push_back(pp);
    }

    int minind=0;
    for(int i=1; i<n; i++)
        if(p[i].y<p[minind].y) minind=i;

    for(int i=1; i<n; i++)
        if(p[i].y==p[minind].y)
            if(p[i].x<p[minind].x) minind=i;

    swap(p[0], p[minind]);
    sort(p.begin()+1, p.end(), cmp);
    p[n]=p[0];

    int k=2;
    for(int i=3; i<=n; i++)
    {
        while(det(p[k], p[k-1],p[i])>=0) k--;
        k++;
        swap(p[i], p[k]);
    }

    n=k;
    view();

    return 0;
}
```

3.3 Сложност на алгоритъма

Алгоритъмът *Graham Scan* на *Роналд Грейм* за изпъкнала обвивка има сложност около O($n$log$n$), която е най-оптималната сложност за такъв алгоритъм.

В таблицата по-долу можем да видим всички алгоритми за изпъкнала обвивка, измислени до момента, и техните сложности и създатели.

Алгоритъм	Сложност	Създател
Brute Force	O(n^4)	Anon
Gift Wrapping	O(nh)	Chand и Kapur
Jarvis March	O(nh)	Jarvis
QuickHull	O(nh)	Eddy и Bykat
Graham Scan	**O($n$log$n$)**	**Ronald Graham**
Divide-and-Conquer	O($n$log$n$)	Preparata и Hong
Monotone Chain	O($n$log$n$)	Andrew
Incremental	O($n$log$n$)	Kallay
Marriage-before-Conquest	O($n$log$h$)	Kirkpatrick и Seidel

4 Алгоритъм за постряване на изпъкнала спирала (авторски)

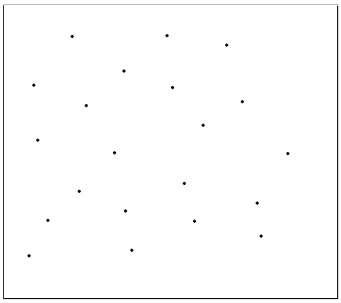
Разработихме наш алгоритъм за намиране на *изпъкналата спирала* за дадено множество от точки в равнината на базата на алгоритъма за строене на изпъкнала обвивка *Graham Scan*, като го повтаряме, докато не останем с една неизползвана точка от множеството.

Пример за стъпките, по които строим изпъкнала спирала, можем да видим на фигури 5а-д.

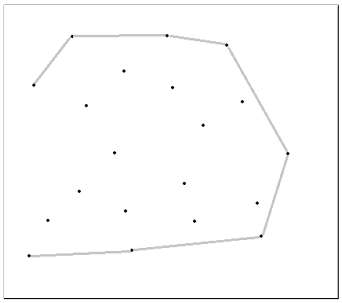
4.1 Стъпки

Началото на координатната система, която използваме, се намира в горния ляв ъгъл на изображението.

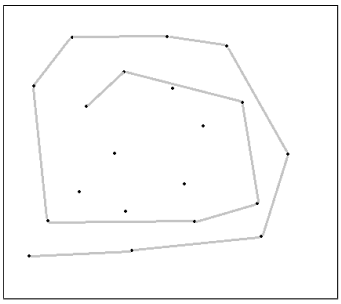
- Намираме най-долния пиксел (работим с всяка намерена звезда чрез пиксел от нея, с координати, запазени някъде във вектора *spiral*), т.е. този с максимална *y координата* (използваме координатна ситема, която започва от началото на изображението). Ако има повече от един такъв пиксел, избираме най-лявата от тях, т.е. тази с минимална *x координата*.
- Разменяме първата точка с намерената.
- Сортираме останалите точки по ъглите на всяка спрямо първата.



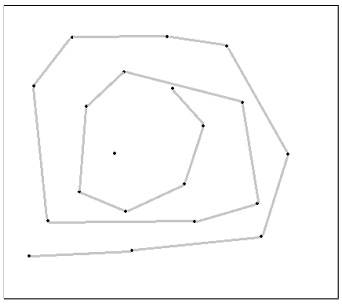
(а) Строене на изпъкнала спирала - етап 1



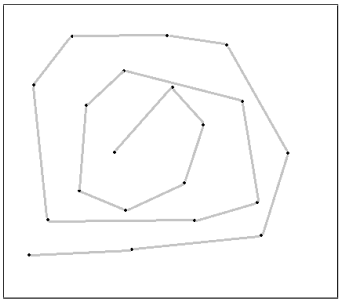
(б) Строене на изпъкнала спирала - етап 2



(в) Строене на изпъкнала спирала - етап 3



(г) Строене на изпъкнала спирала - етап 4



(д) Строене на изпъкнала спирала - етап 5

- Добавяме за последна точка такава като първата.
- Запчваме обхождане на точките от третата до последната точка.

  Всеки път проверяваме дали предните две намерени точки от обвивката и текущата лежат на една права или детерминантата им е положителна.

  Когато намерим такава точка, разменяме следващата с текущата в последовтаелността.

  При всяка размяна увеличаваме броячът на елементи от изпъкна-

лата обвивка k, който в началото има стойност 2 (започваме от третия елемент).

- Накрая в *spiral* получаваме координатите на пикселите, в реда, в който трябва да ги свържем, за да получим изпъкналата обвивка.

4.1.1 Реализация на *C++* в програмата ни

```
void MakeSpiral()
{

    spiral.clear();

    int maxind=0;

    for(int i=1; i<points.size(); i++)

        if(points[i].y > points[maxind].y) maxind=i;

    for(int i=1; i<points.size(); i++)

        if(points[i].y == points[maxind].y) if(points[i].x < points[maxind].x) m

    swap(points[0], points[maxind]);

    spiral.push_back(points[0]);

    for(; ;)

    {

        if(points.size()==1 || points.size()==0) break;


        sort(points.begin()+1, points.end(), cmp);

        points.push_back(points[0]);


        int k=2;

        for(int i=3; i<points.size(); i++)

        {
```

```
            while(det(points[k], points[k-1], points[i])>0) k--;
            k++;
            swap(points[i], points[k]);
        }

        for(int i=1; i<k; i++) spiral.push_back(points[i]);
        points.erase(points.begin(), points.begin()+(k-1));
        points.erase(points.begin()+1, points.begin()+2);
    }
}
```

5 Превръщане на астрономическа снимка в монохромно изображение (авторски алгоритъм)

Разработихме наш алгоритъм за превръщане на астрономическа снимка в монохромно изображение (изображение, състоящо се само от черни и бели пиксели), като намираме стойностите на *R (Red), G (Green) и B(Blue)* на всеки пиксел от снимката и на базата на това определяме дали трябва да го направим бял или черен. С експериментиране открихме каква е границата - пикселите със стойности на цветовете от 0 до 139 са черни, а останалите – бели.

5.1 Стъпки

- Задаваме стойността на *итератора iter* да сочи към снимката, която сме отворили в прозореца на програмата.
- Започваме обхождане на всички пиксели на снимката чрез *iter*.

  iter.r(), *iter.g()* и *iter.b()* показват стойности съответно на червеното, зеленото и синьото от разглеждания пиксел.

  Проверяваме дали *iter.r()*, *iter.g()* и *iter.b()* на раглеждания пиксел са между 0 и 139, т.е. дали пикселът е достатъчно тъмен.

  Ако това е така, определяме, че пикселът е черен.

  Иначе определяме, че пикселът е бял.
- Функцията ни връща вече обработената снимка.

5.2 Реализация на *C++* в програмата ни

```
Surface::Iter TurnBinary( Surface *surface, Area area )
{
    Surface::Iter iter = surface->getIter( area );

    while( iter.line() )
    {
        while( iter.pixel() )
        {
            if((iter.r()>=0 && iter.r()<=139) && (iter.g()>=0 && iter.g()<=139) &
            {
                iter.r() = 0;
                iter.g() = 0;
                iter.b() = 0;
            }
            else
            {
                iter.r() = 255;
                iter.g() = 255;
                iter.b() = 255;

            }
        }
    }

    return iter;
}
```

5.3 Пример

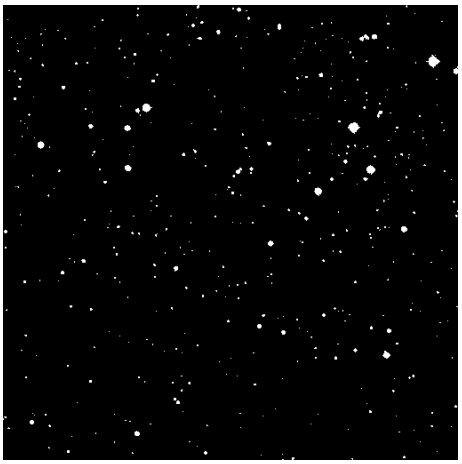
Реализираме алгоритъма чрез функция, написана на *C++* с използването на библиотеката *Cinder*. Тя приема астрономическа снимка и връща монохромно изображение (изображение, състоящо се само от черни и бели пиксели).

Пример за преобразуване на астрономическа снимка в двоично изображение виждаме на фигура 4а и 4б.

6 Определяне на един пиксел от всеки бял фрагмент (авторски алгоритъм)

Разработихме наш алгоритъм за превръщане на всеки бял фрагмент от двоично изображение в един червен пиксел. Определяме тези пиксели, като проверяваме дали всеки бял пиксел дали има черен отляво, отгоре и отдясно над него.

(а) Примерно входно астрономическо изображение



(б) Примерно резултатно двоично изображение

6.1 Стъпки

Началото на координатната система, която използваме, се намира в горния ляв ъгъл на изображението.

- Задаваме стойността на *итератора iter* да сочи към снимката, която сме отворили в прозореца на програмата.
- Започваме обхождане на всички пиксели на снимката чрез *iter*.

iter.r(), *iter.g()* и *iter.b()* показват стойности съответно на червеното, зеленото и синьото от разглеждания пиксел.

Ако текущия пиксел е бял, то

Ако е първият пиксел от изображението (с *координати x=0 и y=0*), добавяме координатите му към вектора *points* .

Иначе, ако се намира в първия ред от пиксели на изображението (с *координата y=0*), ако съседният му ляв пиксел е черен (с отместване -1 по x и 0 по y *координатата*), добавяме координатите му към вектора *points*.

Иначе, ако е първият пиксел от ред от пиксели на изображението (с *координата x=0*), ако съседният му горен пиксел(с отместване 0 по x и -1 по y *координатата*) е черен, добавяме координатите му към вектора *points*.

Иначе, ако е последният пиксел от ред от пиксели на изображението (с *координата x=дължината на реда-1*), ако съседният му горен и съседният му ляв пиксел са черни (съответно с отместване 0

по x и -1 по y *координатата* и отместване -1 по x и 0 по y *координатата*), добавяме координатите му към вектора *points*.

Иначе, ако съседният му горен, съседният му ляв и горният му десен пиксел са черни (съответно с отместване 0 по x и -1 по y *координатата*, отместване -1 по x и 0 по y *координатата* и отместване 1 по x и -1 по y *координатата*), добавяме координатите му към вектора *points*.

6.2 Реализация на *C++* в програмата ни

```
void TurnPixel( Surface *surface, Area area )
{
    Surface::Iter iter = surface->getIter( area );

    while( iter.line() )
    {
        while( iter.pixel() )
        {
            point tmppoint;
            tmppoint.x=iter.x(); tmppoint.y=iter.y();
            tmppoint.number=num+1;

            if(iter.r()==255 && iter.g()==255 && iter.b()==255)
            {
                if(iter.x()==0 && iter.y()==0)
                {
                    iter.r() = 15;
                    iter.g() = 210;
                    iter.b() = 100;

                    num++;

                    points.push_back(tmppoint);
                }

                else if(iter.y()==0)
                {
                    int prevx=-1;
                    int prevy=0;

                    if(iter.r(prevx, prevy)==0 && iter.g(prevx, prevy)==0 && ite
                    {
                        iter.r() = 15;
                        iter.g() = 210;
                        iter.b() = 100;
```

```
            num++;

            points.push_back(tmppoint);
        }
    }
    else if(iter.x()==0)
    {
        int upx=0;
        int upy=-1;

        if(iter.r(upx, upy)==0 && iter.g(upx, upy)==0 && iter.b(upx,
        {
            iter.r() = 15;
            iter.g() = 210;
            iter.b() = 100;

            num++;

            points.push_back(tmppoint);
        }
    }
    else if(iter.x()==iter.getWidth())
    {
        int prevx=-1;
        int prevy=0;

        int upx=0;
        int upy=-1;

        if((iter.r(upx, upy)==0 && iter.g(upx, upy)==0 && iter.b(upx
        {
            iter.r() = 15;
            iter.g() = 210;
            iter.b() = 100;

            num++;

            points.push_back(tmppoint);
        }
    }
    else
    {
        int upx=0;
        int upy=-1;
```

```
                    int prevx=-1;
                    int prevy=0;

                    int up2x=1;
                    int up2y=-1;

                    if((iter.r(upx, upy)==0 && iter.g(upx, upy)==0 &&
                    iter.b(upx, upy)==0) && (iter.r(prevx, prevy)==0 && iter.g(p
                    {
                        iter.r() = 15;
                        iter.g() = 210;
                        iter.b() = 100;

                        num++;

                        points.push_back(tmppoint);
                    }
                }
            }
        }
    }
}
```

7 Алгоритъмът на Брезенхам за строене на отсечка между два пиксела

Използвахме *алгоритъмът на Брезехам*, за да строим бели отсечки между пикселите, които открихме. [15]

7.1 Реализация на *C++* в програмата ни

```
void MakeLine(Surface *surface, Area area, int x1, int y1, int x2, int y2)
{
    Surface::Iter iter = surface->getIter( area );
    int x, y, dx, dy, dx1, dy1, px, py, xe, ye, i;

    dx=x2-x1; dy=y2-y1;
    dx1=fabs(dx); dy1=fabs(dy);
    px=2*dy1-dx1; py=2*dx1-dy1;

    if(dy1<=dx1)
    {
        if(dx>=0)
        {
            x=x1; y=y1;
```

```
            xe=x2;
        }
        else
        {
            x=x2; y=y2;
            xe=x1;
        }

        iter.r(x, y) = 255;
        iter.g(x, y) = 255;
        iter.b(x, y) = 255;

        for(i=0;x<xe;i++)
        {
            x=x+1;
            if(px<0)
            {
                px=px+2*dy1;
            }
            else
            {
                if((dx<0 && dy<0) || (dx>0 && dy>0))
                {
                    y=y+1;
                }
                else
                {
                    y=y-1;
                }
                px=px+2*(dy1-dx1);
            }

            iter.r(x, y) = 255;
            iter.g(x, y) = 255;
            iter.b(x, y) = 255;
        }
    }
    else
    {
        if(dy>=0)
        {
            x=x1;
            y=y1;
            ye=y2;
        }
        else
```

```
            {
                x=x2;
                y=y2;
                ye=y1;
            }

            iter.r(x, y) = 255;
            iter.g(x, y) = 255;
            iter.b(x, y) = 255;

            for(i=0;y<ye;i++)
            {
                y=y+1;
                if(py<=0)
                {
                    py=py+2*dx1;
                }
                else
                {
                    if((dx<0 && dy<0) || (dx>0 && dy>0))
                    {
                        x=x+1;
                    }
                    else
                    {
                        x=x-1;
                    }
                    py=py+2*(dx1-dy1);
                }

                iter.r(x, y) = 255;
                iter.g(x, y) = 255;
                iter.b(x, y) = 255;
            }
        }
}
```

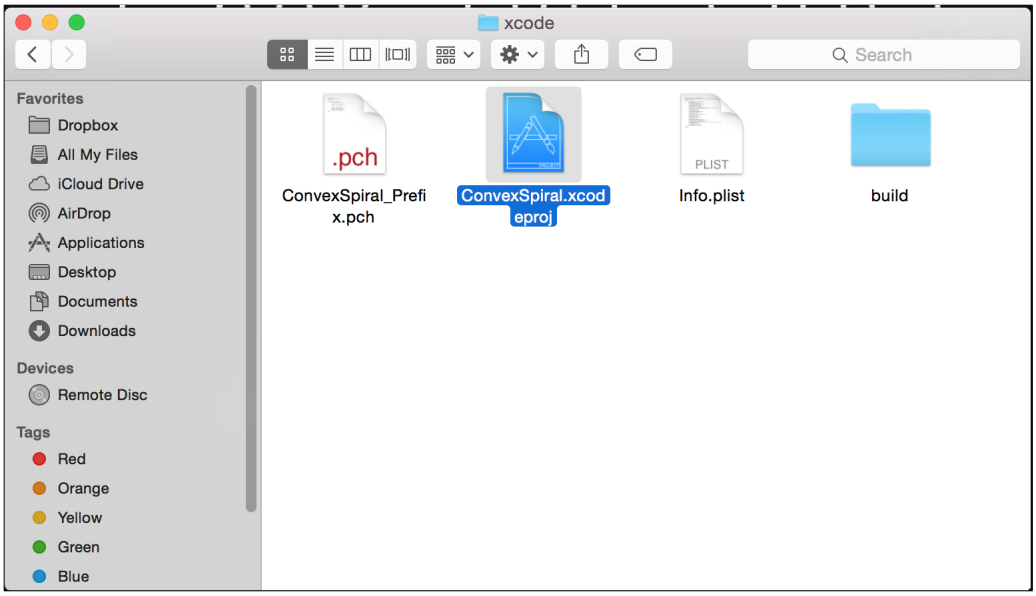
8 Работа с пректа

В програмата изпълняваме последователно алгоритмите, с които се запознахме. Получихме приложение, работещо на *Mac OS X* и *Windows*.

Трябва да разполагаме с библиотеката *Cinder* и блоковете й за *OpenGL*, за да работим с проекта.[17]

Първо, отваряме *ConvexSpiral* - папката на проекта ни. След това, ако операционната ни ситема е *Windows*, отваряме папката *vc2013*. Трябва ни *Visual Studio 2013*, за да работим с *ConvexSpiral.vcxproj*. Иначе, ако опера-

ционната ни система е *Mac OS X*, отваряме папката *xcode*. Трябва ни *X Code*, за да работим с *ConvexSpiral.xcodeproj* (фигура 5).



Фигура 5: Отваряме проекта

Вече можем да разгледаме и редкатираме кода на проекта ни (фигура 6).

Можем да компилираме проекта. (фигура 7) Ако не успеем, трябва да проверим дали в *Resources.h* в папката *include* на проекта е посочен точният път към папката на библиотеката *Cinder* на нашия компютър.

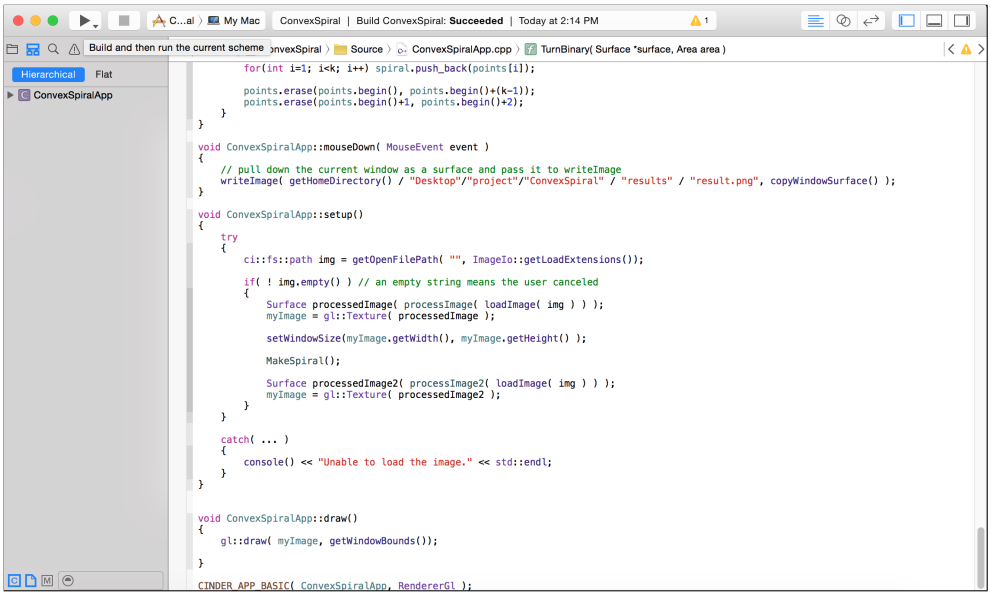
След като компилираме успешно проекта, трябва да изберем снимка в *JPEG* формат, върху която да построим изпъкнала спирала (фигура 8).

След това, прозорецът на проекта ще показва снимката с нарисуваната върху нея спирала (фигура 9).

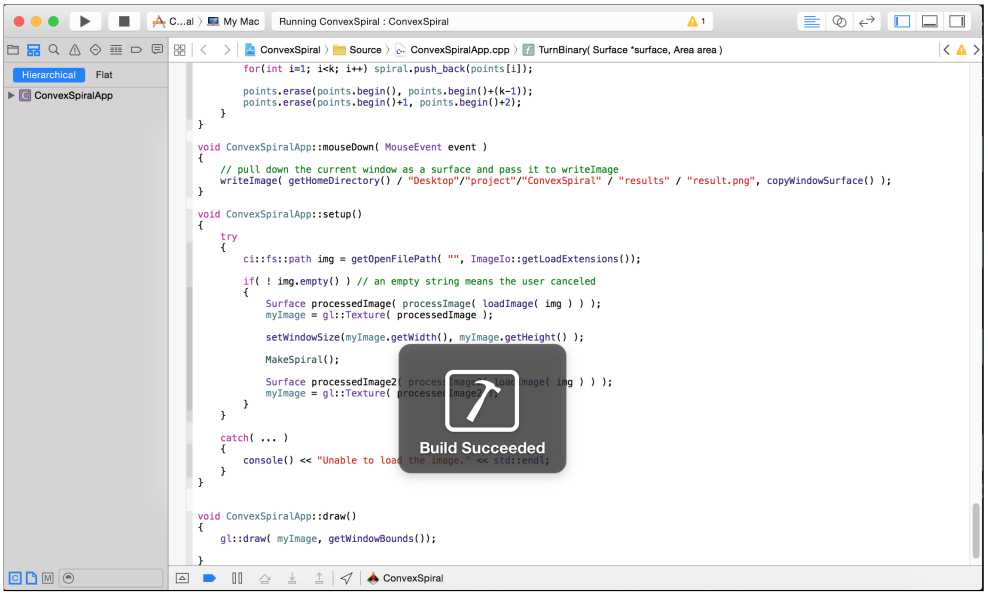
Можем да запазим изображението, като щракнем върху прозореца. Трябва да посочим точна директория във функцията *mouseDown(MouseEvent event)*, където да се запази. (фигура 10)

9 Примери

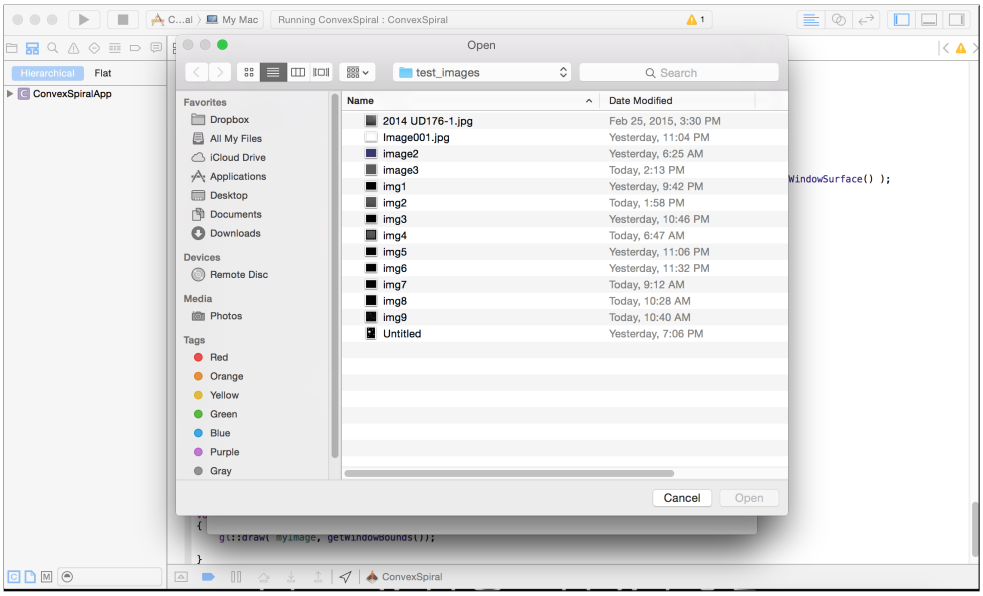
Виждаме примери за изпъкнали спирали на различни астрономически снимки на фигури 11а, 11б, 12а, 12б, 13а, 13б, 14а, 14б.



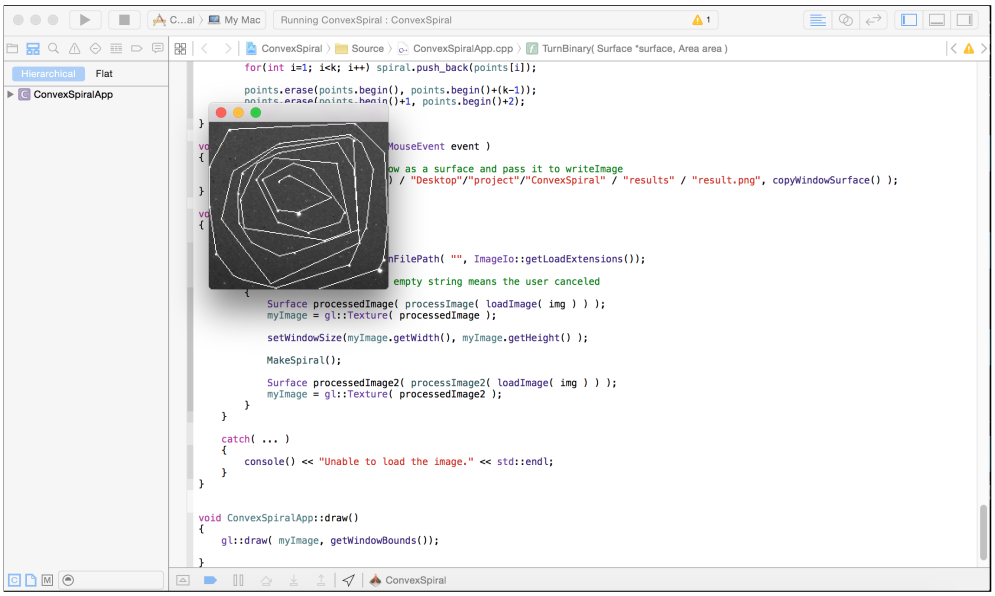
Фигура 6: Кодът на проекта



Фигура 7: Кодът на проекта



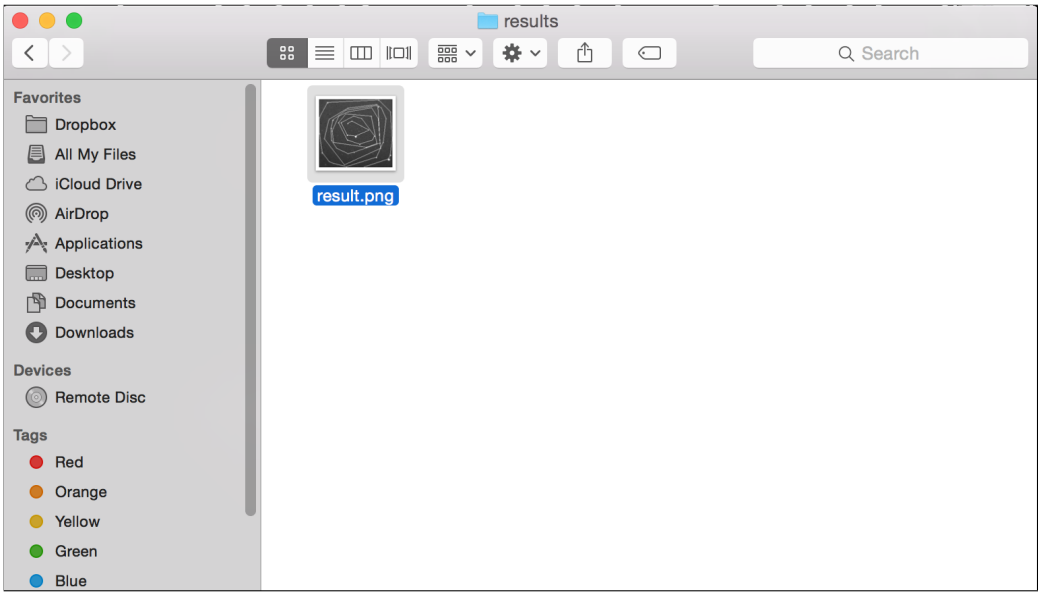
Фигура 8: Отваряне на снимка



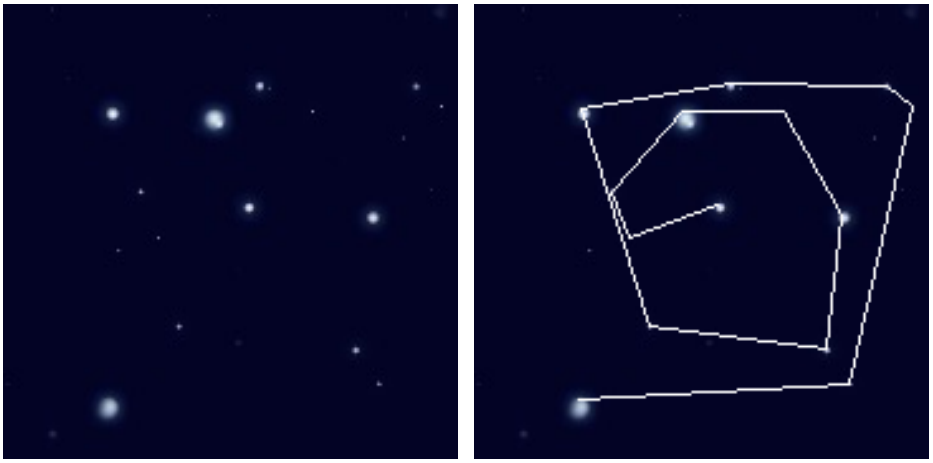
Фигура 9: Резултатно изображение

10 Потребители

Проектът е полезен както за професионалисти, така и за любители-астрономи. Заради формата за приемане на снимките - JPEG, няма нужда да притежа-



Фигура 10: Запазеното изображение



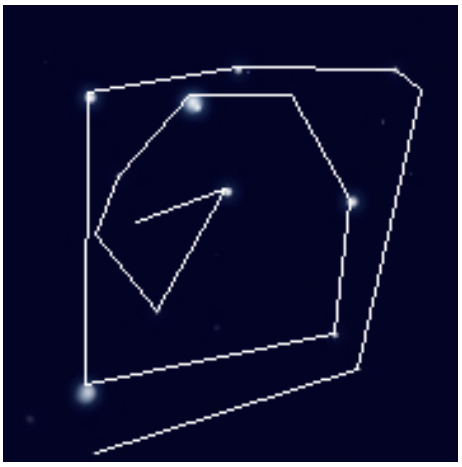
(а) Входно изображение (б) Резултатна спирала

ват телескоп, могат и сами да заснемат небето. Въпреки това, за да получат най-добри резултати, най-подходящо е да работят с професионални данни.

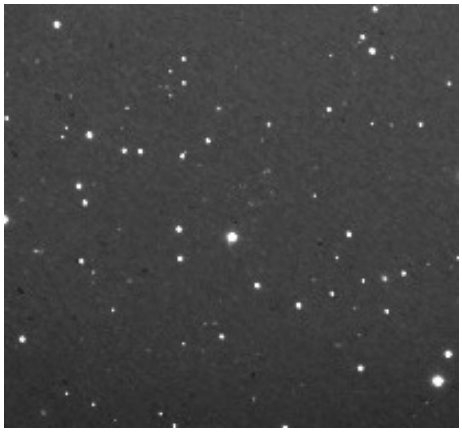
Нашият проект ще е полезен за астроклуба на *НАОП „Николай Коперник“* във Варна. Там, с помощта на *Astrometrica*, се търсят астероиди от снимки от телескоп на *НАО – Рожен*.

IASC – International Astronomical Search Collaboration [9] е сътрудничество между хора от цял свят (най-често от астроклубове на обсерватории),

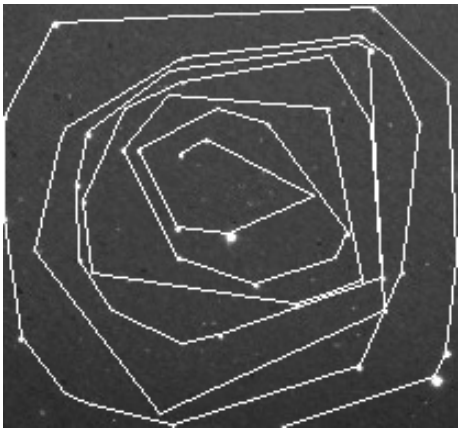
(а) Входно изображение



(б) Резултатна спирала



(а) Входно изображение



(б) Резултатна спирала

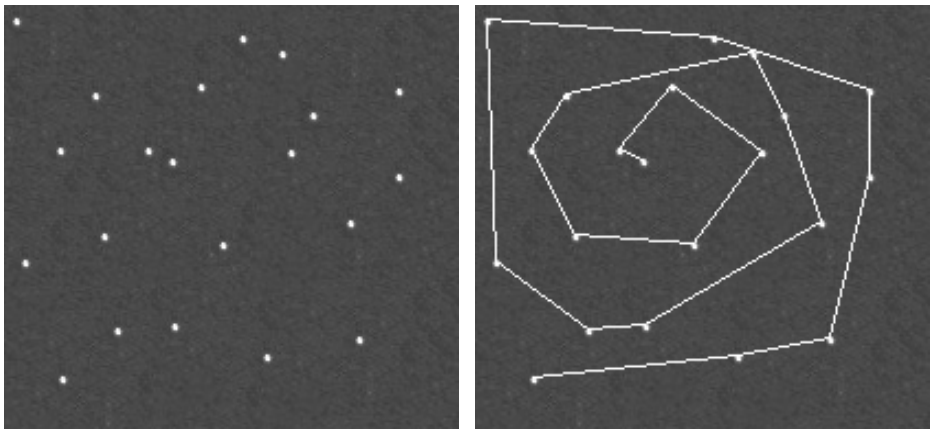
които търсят движещи се обекти, които са най-често астероиди, от астрономически снимки, с помощта на програми, помагащи за анализирането на тези данни.

Нашият проект ще е много полезен за развитието на тази кампания, като по този начин също ще се лансира новия метод за сравнение на астрономически изображения.

11 Проблеми

Сблъскахме се с няколко проблема докато разработвахме приложението.

Един от тях беше свързан с директориите. Трябаше да настроим пътя към библиотеки в папката include/Resources.h, така че да отговаря на



(а) Входно изображение (б) Резултатна спирала

нашата компютърна папка, за да можем да работим с Cinder.

Вторият беше, че алгоритъмът ни работеше само при ?Кортезианова? координатна система (такава, която започва от центъра на изображението), но се оказа, че трябва да използваме такава, която започва от началото на изображението. Променихме операциите с *y координати* в алгоритъма си, за да работи в тези условия.

Друг проблем беше, че отсечките между пиксели, построени чрез функцията на emph$OpenGL$ - GL_LINES не винаги са точни. Открихме, че с този проблем се бяха сблъсквали и други. [16] Затова използвахме алгоритъма на Брезенхам за построяване на такива отсечки.

12 Заключение

В разработката описахме основните понятия, свързани с компютърната обработка на астрономически изображения, както и езиците за програмиране, на които написахме програмите от проекта.

Разгледахме алгоритъма *Graham Scan* за строене на изпъкнала обвивка на множество от точки в равнината, неговата реализация на езика *C++* и сложността му в сравнение с други съществуващи такива алгоритми.

Разработихме софтуер, който строи *изпъкнала спирала* по астрономическа снимка. Написахме програма на *C++* в комбинация с библиотеката *Cinder*, която работи на компютри с *Windows* и *Mac OS X*. Тя приема астрономическа снимка във формат *JPEG*. След това я преобразува в монохромно изображение, тоест изображение, съдържащо само черни и бели пиксели. Открива по един пиксел за всеки бял фрагмент и запазва координатите му. Чрез собствен алгоритъм за намиране на *изпъкналата спирала* на можество от точки в равнината, подрежда пикселите в реда, в който трябва да ги свържем, за да получим спиралата. След това ги свързва тези

пиксели чрез алгоритъма на Брезенхам за растеризация на отсечка. Получаваме нарисувана върху снимката *изпъкнала спирала*.

В бъдеще ще направим сайт, в който астрономи да могат да използват проекта ни. Планираме да подобрим алгоритъмите за намиране на светли обекти в снимка. Предвиждаме и създаването на програма, която да сравнява изпъкналите спирали, получени от последователно взети астрономически снимки, като по този начин улеснява намирането на астероиди.

Ефективността на инструменти като *Astrometrica* сочи, че софтуер за сравняване на астрономически снимки е както нужен, така и особено полезен, тъй като засяга една от най-важните теми в астроинформатиката сега – търсенето на астероиди и други движещи се обекти по снимки.

Литература

[1] Patrick Miller *Instructions for Using Astrometrica* September 2008

[2] Herbert Raab *Detecting and measuring faint point sources with a CCD* Astronomical Society of Linz, Sternwarteweg 5, A-4020 Linz, Austria

[3] Justin Iwerks and Joseph S. B. Mitchell *Spiral serpentine polygonization of a planar point set* 11794-3600: Dept. of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY

[4] Wan D. Bae, Shayma Alkobaisi, Sada Narayanappa, Petr Vojtechovsky, and Kye Y. Bae *Optimizing map labeling of point features based on an onion peeling approach* August 2, 2010: Department of Mathematics, Statistics, and Computer Science, University of Wisconsin-Stout, Menomonie, WI 54751, USA

[5] M. Loffler and W. Mulzer. *Unions of Onions: Preprocessing Imprecise Points for Fast Onion Decomposition* Proc. 13th WADS, pp. 487–498, 2013

[6] Mohammad Reza Razzazi and Ali Sajedi *Kinetic Convex Hull Maintenance Using Nested Convex Hulls* Software Research and Development Laboratory, Computer Engineering Department, AmirKabir University of Technology, Tehran, Iran

[7] Keith S Cover *Improved visual detection of moving objects in astronomical images using color intensity projections with hue cycling* 2012, VU University Medical Center Amsterdam, The Netherlands

[8] `http://fits.gsfc.nasa.gov/fits_viewer.html`

[9] `http://iasc.hsutx.edu/`

[10] `http://astro-varna.com/astroclub/`

[11] `http://iasc.hsutx.edu/iasc/TA!.html`

[12] `http://www.faulkes-telescope.com/files/faulkes-telescope.com/archive/activities/introductory_activities/software/Astrometrica_asteroids.pdf`

[13] `http://cgm.cs.mcgill.ca/~orm/sptri.html`

[14] `http://math.stackexchange.com/questions/410602/what-is-the-expected-convex-depth-of-a-set-of-m-randomly-chosen-points-in-the`

[15] `http://cpp-examples.com/singlelecture.php?id=rasterization_of_the_edge`

[16] `http://stackoverflow.com/questions/14011588/gldrawarrays-gl-lines-not-drawing-lines-correctly`

[17] `http://libcinder.org`