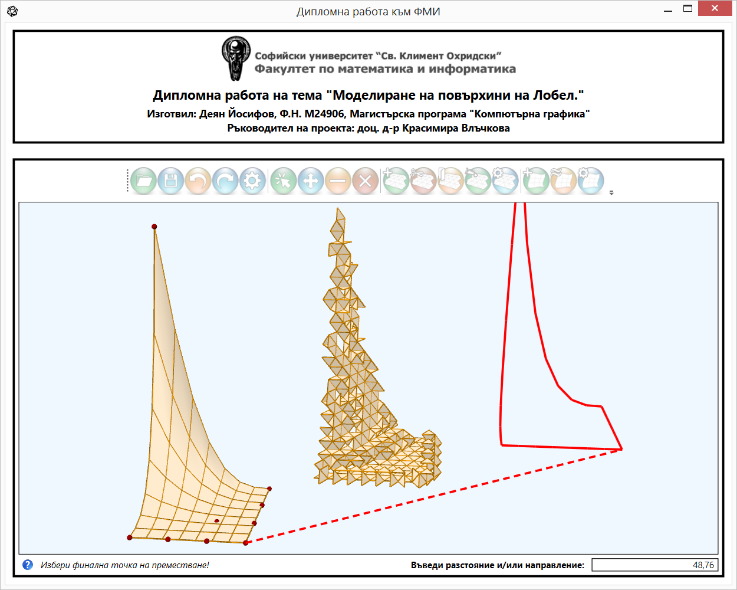


**Моделиране на повърхнини на Лобел**

****

**Cъдържание**

1. [Увод](#_Увод)
   1. [Проблеми на „Не-Евклидовата“ архитектура](Не-Евклидовата#_Проблеми_на_)
   2. [Повърхнини на Лобел](#_Повърхнини_на_Лобел)
   3. [Описание на поставените проблеми](#_Описание_на_поставените)
2. [Математически метод и алгоритми](#_Математически_метод_и)
   1. [Моделиране на принципа на Киригами](#_Моделиране_на_принципа)
      1. [Рязане](#_Рязане)
      2. [Прегъване](#_Прегъване)
      3. [Прегъване със залепяне](#_Прегъване_със_залепяне)
      4. [Залепяне](#_Залепяне)
   2. [Моделиране с апроксимиращи алгоритми](#_Моделиране_с_апроксимиращи)
      1. [Дискретна UV мрежа](#_Дискретна_UV_мрежа)
      2. [Окта-Тетра мрежа](#_Окта-Тетра_мрежа)
      3. [Ориентирана проекционна площ](#_Ориентирана_проекционна_площ)
      4. [Ориентиран проекционнен обем](#_Ориентиран_проекционен_обем)
      5. [Усреднено ориентирано проекционно разстояние](#_Усреднено_ориентирано_проекционно)
      6. [Апроксимиращи алгоритми – обща информация и начални условия](#_Апроксимиращи_алгоритми_–)
      7. [Lobel Mesh Projecting Algorithm](#_Lobel_Mesh_Projecting)
      8. [Centroids Distance Measuring Algorithm](#_Centroids_Distance_Measuring)
      9. [Intersecting Volumes Finding Algorithm](#_Intersecting_Volumes_Finding)
      10. [Intersecting Volumes Connecting Algorithm](#_Intersecting_Volumes_Connecting)
3. [Имплементационни детайли](#_Имплементационни_детайли)
   1. [Използвана технология](#_Използвана_технология)
   2. [Структури от данни](#_Структури_от_данни)
      1. [IMeshElementsProvider](#_IMeshElementsProvider)
      2. [IDescreteUVMesh](#_IDescreteUVMesh)
      3. [IBezierMesh](#_IBezierMesh)
   3. [Команди](#_Команди)
   4. [Файлови формати](#_Файлови_формати)
      1. [OBJ файлов формат](#_OBJ_файлов_формат)
      2. [LOBZ файлов формат](#_LOBZ_файлов_формат)
4. Помощ за приложението
5. Библиография

# Увод

С напредването на технологиите все по-често забелязваме модерни архитектурни явления, характерни с техните разчупени форми, различаващи се от стандартния за архитектурата паралелепипед. Сред известните имена на архитекти, търсещи нестандартните решения, са Франк Гери и Заха Хадид, но и мнозина други архитекти от нашето съвремие. Често пъти, коректно или не, този тип архитектура бива наречен „Не-Евклидова архитектура“ заради кривите форми на използваните повърхнини. Примери на някои такива проектни решения могат да се видят на следващите картинки.



## Проблеми на „Не-Евклидовата архитектура“

Проектирането и последващото практическо изпълнение на този тип архитектура в повечето случаи струва в пъти повече от стандартната архитектура. Оскъпяването идва от няколко различни фактора:

* **Сложности при самото проектиране**. Моделирането на такива форми изисква по-сложни изчисления, касаещи отделните елементи и съгласуването на инсталации от различни специалности. Отделно изработването на детайли и четими монтажни схеми е нелека задача, тъй като често пъти всеки отделен елемент и детайл е различен от другите.
* **Сложност при изработката на материалите**. Ако формата не бъде разчленена на някакви равнинни парчета, това често пъти прави изработката на отделните елементи доста трудна и скъпа. Дори и отделните парчета да са равнинни, наличието много различни видове форми на всяка от частите също оскъпява и забавя допълнително производствения процес.
* **Сложност при строежа на архитектурния обект**. Сглобяването на всичките различни елементи на сградата може да се окаже доста сложна задача, дори при наличието на добри монтажни чертежи и схеми. Това изисква високо квалифицирана работна ръка, която е по-скъпа. Отделно самият процес е по-бавен от конвенционалното строителство, а както знаем „времето е пари“.

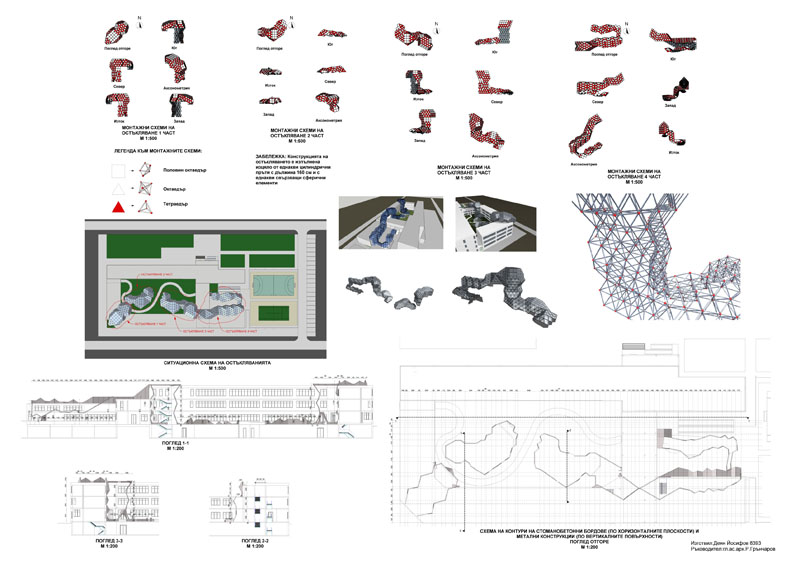
Всички тези оскъпяващи проблеми биха могли да се решат ако сложната структура се изгради от максимален брой еднакви елементи с минимален брой разлики в монтажните детайли между отделните парчета. Едно такова решение са така наречените повърхнини на Лобел, за които следва да обясним какво представляват.

## Повърхнини на Лобел

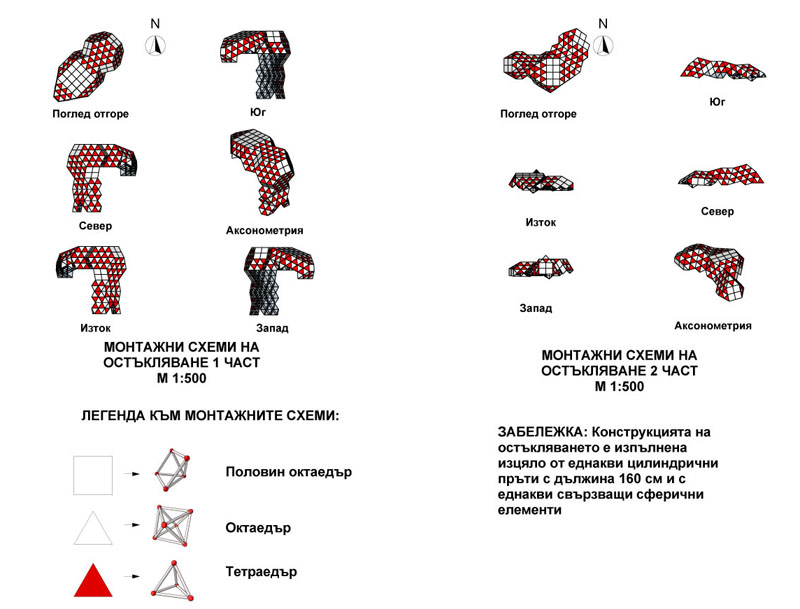
В последните няколко години френският архитект Ален Лобен работи по изследването на архитектурни структури, изградени изцяло от еднакви равностранни триъгълници. Вероятно други хора също са се сблъсквали с този въпрос, но разработките на този френски архитект в неговия web site <http://www.equilatere.net/> постепенно налагат името на такива структури като „Повърхнини на Лобел“ или на английски – „Lobel Frames”.

Самият аз като студент по архитектура през 2011 година разработих преддипломен работен проект на реконструкция на НПМГ „акад. Любомир Чакалов“, който проект използваше метална конструкция, изградена изцяло от еднакви равностранни триъгълници. По това време не бях чувал за разработките на Ален Лобел, макар и да бях търсил доста в интернет като проучване разработки на подобни конструкции. Това ме води до мисълта, че вероятно разработките на френския колега са сравнително скорошни и дори да са били налични през 2011, то те са били твърде нови в интернет пространството, за да бъдат откриваеми.

По-долу може да видите едно от таблата към моя проект, което има ситуация, изгледи, перспективи и монтажни схеми на въпросната конструкция.



Особено внимание бих искал да обърна с по-близък поглед към част от монтажните схеми, които разясняват простотата на въпросната конструкция.



Както може да се види от легендата, цялата конструкция е изградена от еднакви октаедри и тетраедри, които в крайна сметка оформят една повърхнина от еднакви равностранни триъгълници. Тази повърхнина позволява да бъде изградена изцяло от еднакви пръти, свързани в еднакви сферични възли. Еднаквостта на елементите позволява едновременно бързо производство и бърз и лесен монтаж на мястото на обекта, следвайки простите двуцветни монтажни схеми (червен триъгълник означава тетраедър, а бял означава октаедър, както е видно и от легендата).

Единствената трудност, която остана нерешена, е сложността на проектирането на въпросната конструкция. По времето, когато аз работих по този проект, нямаше софтуер, който да ми помага с лесното моделиране на такъв тип конструкция. Това, което направих тогава, беше да напиша един прост plug-in за Google Sketch Up, който да може от наличен равностранен триъгълник да вдига тетраедър, октаедър или икосаедър. Този плъгин може да бъде видян в моя Github акаунт: <https://github.com/deyan-yosifov/Deyan-Projects/tree/master/Ruby/SketchUpPlugins/dpy-equilateral>. Макар и с този плъгин, моделирането на цялостната структура беше доста трудоемка задача. Първоначално измоделирах една крива произволна повърхнина с известна програма, така че формата да отговаря на практическите и визуални цели, които търсех. След това вкарах модела в Google Sketch Up и започнах да добавям и трия тетраедърчета и октаедърчета едно по едно, така че получената структура да доближава първоначално измоделираната повърхнина. Добавянето и триенето на триъгълници един по един беше наистина времеемко, но тогавашните ми слаби познания по програмиране не ми позволиха да автоматизирам този процес.

И тук идва темата на сегашната дипломна работа, целяща да се улесни процеса при моделирането и проектирането на такива структури от еднакви триъгълници.

## Описание на поставените проблеми

Основната цел на сегашната дипломна работа е разработка на софтуер, който да улесни проектирането и моделирането на повърхнини на Лобел. Този софтуер ще предоставя следните възможности:

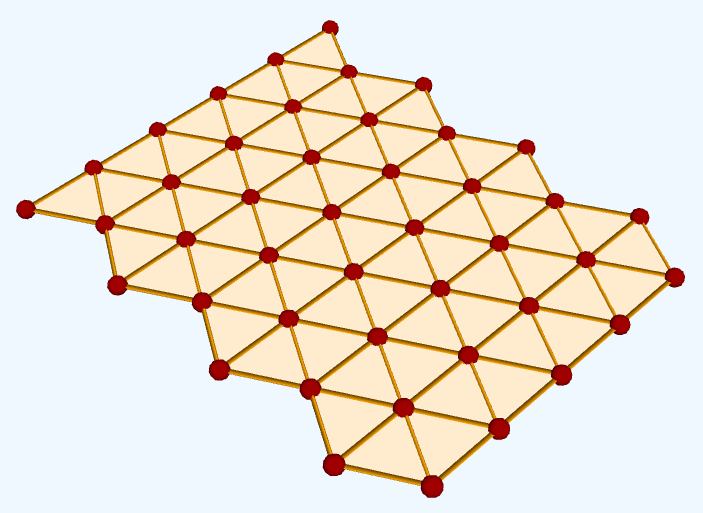
* **Моделиране на повърхнина на Лобел от една равнинна плоскост с равностранни триъгълници на принципа на известното японско изкусктво „киригами“.** На кратко обяснено, киригами представлява рязане, огъване и лепене на лист хартия, така че да се измоделира някаква по-сложна триизмерна структура. На практика всяка повърхнина на Лобел може да се измоделира от един достатъчно голям лист хартия, който е разграфен с мрежа от еднакви равностранни триъгълници. Наличието на операции „рязане“, „огъване“ и „лепене“ на такива разграфени равнинни елементи, би позволило на ползвателя на софтуера да моделира или модифицира дадени повърхнини на Лобел.
* **Предоставяне на алгоритми за приближаване на дадена произволна повърхнина с повърхнина на Лобел.** Докато моделирането със способите на киригами може да се окаже трудоемка и времеемка задача, то предлагането на такива алгоритми би позволило да се измоделира първо една произволна повърхнина и след това с един клик алгоритъмът да предложи приближение на въпросната повърхнина с мрежа от еднакви равностранни триъгълници.
* **Предоставяне на операции за отваряне и записване на работата в определени тримерни файлови формати.** Това е ключово поради три причини. Първо – за да може да моделира обекти от реален проект, ползвателят на софтуера трябва да може да вкара в програмата своя първоначален проект, в който иска да интегрира повърхнината на Лобел. Второ – когато човек използва софтуера, би следвало да има възможност да запише работата си, за да може в последствие да я продължи. Трето – когато работа по моделирането е завършена, тя би следвало да може да бъде записана в известен файлов формат, който да може да бъде отварян от други програми, където работата по моделирането и интегрирането на геометрията може да продължи.
* **Предоставяне на операции за връщане или повтаряне на определена стъпка от работа с програмата.** Тези операции в други програми са известки като Undo-Redo и са ключови от гледна точка, че грешна стъпка по време на моделирането би трябвало да може да бъде лесно върната, а не да трябва да се започва работата от начало.

# Математически метод и алгоритми

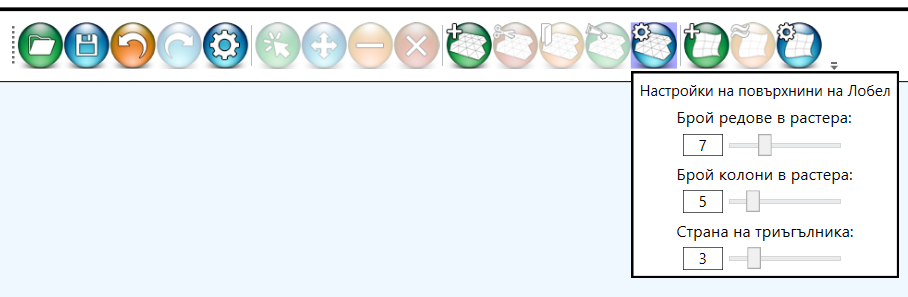
В тази секция ще опиша идеите, реализирани в текущата имплементация на приложението, както и математическите изчисления, необходими за осъществяването на алгоритмите. Както споменахме в уводната част, в приложението ще предложим два подхода за моделиране на повърхнини на Лобел. Първият е моделиране на принципа на Киригами - с рязане, прегъване и лепене на мрежа от еднакви равностранни триъгълници. Вторият метод е с готов алгоритъм, който приближава една произволна 3D повърхнина с повърхнина на Лобел на базата само на зададена страна на равностранните триъгълници от мрежата на Лобел.

## Моделиране на принципа на Киригами

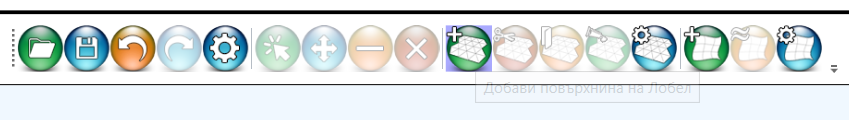
Киригами е известно японско изкуство, което изучава методите на създаване на пространствени макети чрез рязане, прегъване и лепене на лист хартия. На практика всяка повърхнина на Лобел би могла да бъде измоделирана от достатъчно голям лист хартия, разграфен от мрежа с еднакви равностранни триъгълници. Такава мрежа е показана на картинката по-долу.



Както може да се види на долната картинка, мрежа може да се дефинира в приложението с 3 параметъра – брой редове в растера, брой колони в растера и размер на страната на всеки от равностранните триъгълници.



Избирайки желаните параметри, ползвателя на моето приложение е необходимо само да натисне зеления бутон със знак „+“, предвиден за добавяне на нова повърхнина на Лобел.

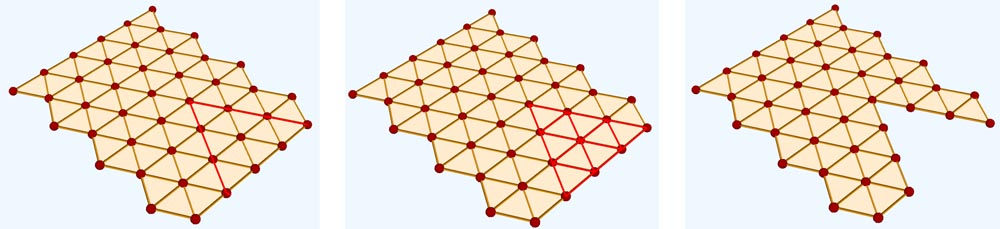


И това е – вече имаме начална мрежа и можем да започнем да я модифицираме с бутоните за рязане, прегъване и лепене, показани по-долу.



#### Рязане

Рязането на мрежата ни позволява да селектираме определени триъгълници и да ги премахнем от съществуващата мрежа. Селекцията се прави чрез избиране на червените точки във върховете на мрежата. На долната картинка са показани последователност от примерно селектиране на точки от мрежата, резултатът от селектираните триъгълници и резултатът след изрязването на въпросните триъгълници.



#### Прегъване

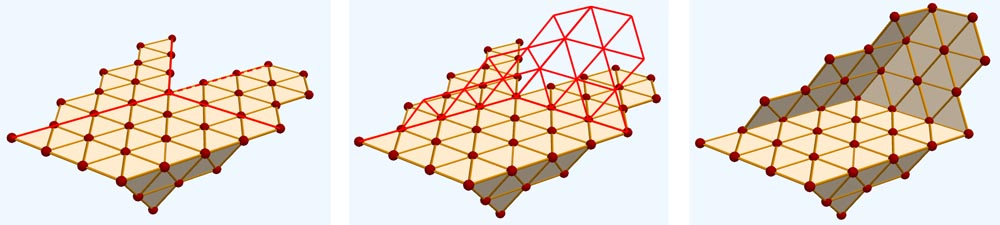
Прегъването е ключова операция, когато искаме равнинната мрежа от равностранни триъгълници да я превърнем в пространствена такава. Едно прегъване може да бъде дефинирано чрез избирането на 3 контролни точки – първите две дефинират оста на ротация (оста на прегъване), а последната трета точка дефинира полуравнината, която ще бъде ротирана (прегъната). На следващите картинки може да се види как примерът от предната секция за „Рязане“ е продължен с една операция за прегъване. Трите съседни картинки показват трите етапа на операцията „Прегъване“ в приложението. Първо се селектират трите контролни точки дефиниращи прегъването. След това се вижда селекцията от триъгълничета, които ще се ротират около оста на прегъване. На последната картинка се вижда резултатът от прегъването след като е избран ъгъл на ротиране 110 градуса.



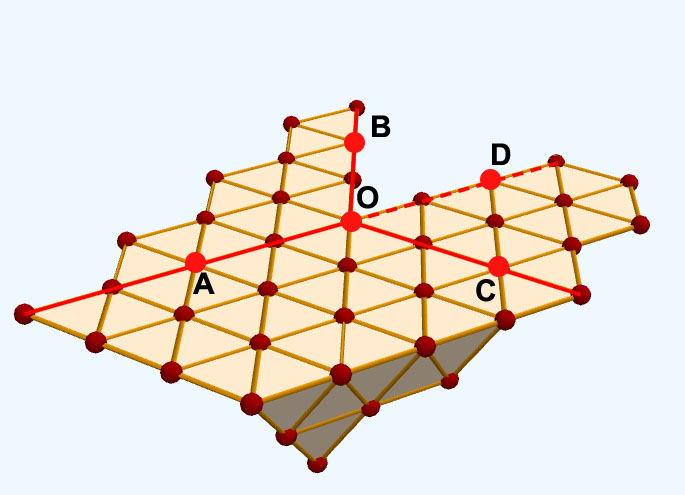
#### Прегъване със залепяне

Често пъти при моделирането на повърхнината може да се наложи да огънем две съседни парчета от мрежата, така че два от ръбовете им да се залепят един за друг. Такова залепяне може да се случи само при определени ъгли на огъване на съответните парчета и тези ъгли на огъване не са лесни за пресмятане с просто око. Поради тази причина моето приложение предлага още един подход за огъване, при който се задават две оси на ротация (на прегъване) и приложелнието пресмята и предлага възможните ъгли на ротация, така че съседните ръбове да се залепят един за друг.

Нека покажем нагледно какво се има предвид под „прегъване със залепяне“, продължавайки моделирането на примера от предните две секции („рязане“ и „прегъване“). За целта ще прегънем двете парчета около дупката, получена след като изпълнихме операцията „рязане“. На долната схема се виждат трите етапа от двуосовото прегъване, които са резултат от операцията в моето приложение. Първият етап е селектирането на 5 контролни точки от мрежата – точка, обща за двете оси на прегъване; точка, дефинираща първата ос на прегъване; точка, дефинираща първата полуравнина на прегъване; точка, дефинираща втората ос на прегъване и точка, дефинираща втората полуравнина на прегъване. На втората картинка се виждат селектираните триъгълници в първото им възможно положение на ротация. Приложението прелага възможност за избор между всички възможни ъгли на ротации чрез кликане с мишката. В случая има две възможни ротации, така че съседните ръбове да съвпадат. Едната е показаната на картинката и при нея ротираните точки са в горното полупространство спрямо първоначалната им равнина. Втората е огледална в долното полупространство. На последната картинка се вижда и резултатът от изпълнението на операцията „прегъване със залепяне“.



Тъй като тази операция изисква с една идея по-сложни изчисления от предните две, ще споменем накратко метода за пресмятане на възможните ъгли на ротация. За целта ще вземем няколко точки от ключовите за операцията линии. Нека точка **О** бъде общата точка за двете оси на ротация. Нека вземем точка **А**, която да лежи на първата ос на ротация и точка **C**, която да лежи на втората ос на ротация. Освен това нека изберем по една точка на всеки от ръбовете, които искаме да бъдат „залепени“ след прегъването. Нека точка **B** бъде върху първия ръб , а точка **D** бъде върху втория ръб. Нека също така точките **B** и **D** да са избрани така, че да са на равно разстояние от точка **О**. По този начин целта на операцията е да намерим ъглите, на които трябва да се ротират парчетата **AOB** и **COD**, съответно около осите OA и OC, така че в крайна сметка след ротациите точките **B** и **D** да съвпаднат. Обозначенията на точките могат да се видят на долната картинка.



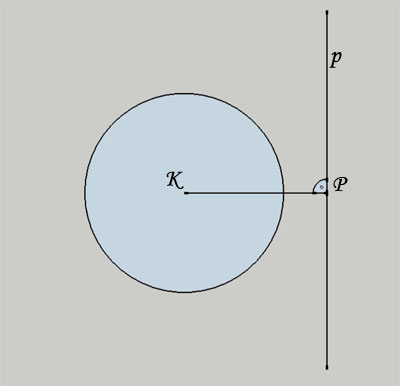
Нека обозначим с **i** и **j** съответно единичните вектори по двете оси, както следва:



При прегъването точките **B** и **D** описват две окръжности. Нека окръжността, описана от точка **B**, е с център **K**, а окръжността, описана от **D**, е с център **L**. Центровете на окръжностите лежат върху съответните оси на ротация. Освен това равнините на окръжностите са с нормали съответно векторите **i** и **j**. Следователно центровете могат да се намерят с проектиране на точките **B** и **D** върху осите на ротация, както следва.



За да намерим възможните ъгли на ротация, така че точките **B** и **D** да съвпаднат, ние на практика трябва да търсим пресечни точки на двете окръжности. Такива точки, ако евентуално съществуват, ще се намират върху пресечната права на равнините на двете окръжности. Нека тази пресечна права между две равнини означим с буквата **p**. Нека също така означим върха на перпендикуляра от центъра **K** до правата **p** с точката **P**. Така отсечката **KP** ще е най-късата, свързваща центъра **K** с правата **p**. Ако разгледаме равнината на окръжността с център K, то ще получим картинка, сходна на показаната по-долу.



Нека видим как можем да пресметнем точката **P**. Кръщаваме единичен вектор по правата **p** със същата буква **p**. Тъй като този вектор лежи едновременно в равнините на двете окръжности, то за него можем да кажем следното.



Тъй като **p** е перпендикулярен и на двете оси на ротация, то той може да бъде сметнат като тяхно векторно произведение.



От своя страна отсечката **KP** е перпендикулярна едновременно на оста на ротация и на правата **p**, откъдето следва, че можем да намерим вектор, колинеарен с отсечката **KP** като пресметнем векторното произведение на **p** и **i**.



Последното условие ни позволява много лесно да пресметнем точката **P** като пресечна точка на права и равнина. А именно, ако дефинираме правата с точка **K** и вектор, колинеарен на **KP** и равнината с точка центъра **L** и нормала вектора **j**, то не е трудно да преценим, че пресечната точка **P** на дефинираните права и равнина може да се сметне със следното уравнение.



Така, след като успяхме да пресметнем точката P, вече можем да се върнем на нашата задача да намерим дали има подходящи пресичания на окръжността с център **K** и правата **p**. Дали има изобщо пресичания можем да кажем в зависимост от това дали отсечката **KP** e по-голяма, по-малка или равна на радиуса на окръжността. Този радиус е равен на дължината на отсечката **KB**.

**Първият случай** е, в който радиусът е по-малък от отсечката. В този случай окръжността изобщо не пресича правата **p** и следователно нашата задача няма решение.



**Вторият случай** е, в който радиусът е точно равен на **KP**. Тогава потенциално решение на нашата задача е точно точката **P**.



В този случай, за да проверим дали **P** наистина е решение, то трябва да проверим дали разстоянието между **P** и центъра на другата окръжност **L** е равно на радиуса на съответната окръжност, който е **LD**.

Ако **LP** е различно от **LD**, тогава задачата ни няма решение.



Ако обаче **LP** е равно на **LD**, тогава **P** е решение на нашата задача. Това означава, че съществуват ъгли **α** и **β**, при които ротираните съответно точки **B** и **D** ще съвпаднат с точката **P**.



**Третият случай** е, в който радиусът **KB** е по-голям от дължината на **KP**. Тогава имаме две пресичания на окръжността с правата **p** и съответно две потенциални точки решения на нашата задача, които ще кръстим с буквите **M** и **N**.



Нека първо сметнем двете пресичания на окръжността с правата **p**. Тези две точки **M** и **N** могат да бъдат изчислени по следния начин.

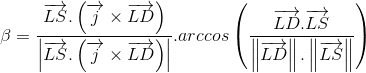
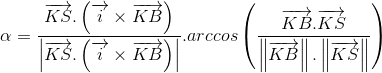


Сега вече можем да проверим дали потенциалните решения **M** и **N** са наистина решения на задачата. Това отново може да бъде направено като бъдат сметнати разстоянията от **M** и **N** до центъра **L** на другата окръжност. Ако разстоянието е равно на радиуса на втората окръжност, то тогава съответната точка може да се каже, че е решение на нашата задача.



Иначе казано, задачата за намиране на възможните решения, така че двата ръба да съвпаднат след прегъването, би могло да доведе до максимално две решения.

За пълнота на пресмятанията ще добавя и последните сметки от приложението, които ще ни позволят да изчислим ъглите на ротация, така че двата ръба да „залепнат“ след прегъването. Нека кръстим с **S** точката - решение на нашата задача, в която точките B и D ще съвпаднат след прегъването. Както вече знаем, S лежи едновременно на двете окръжности. Долните изчисления показват как да намерим съответните ъгли на ротация **α** и **β**, на които трябва да се ротират съоветно точката **B** и точката **D**.

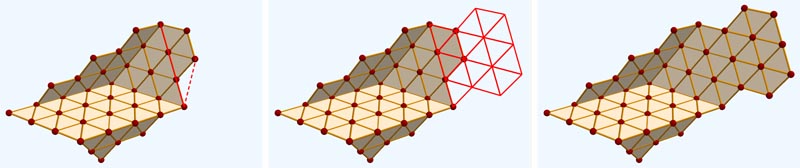


Така, смятайки ъглите на ротация за различните решения на задачата можем да ротираме съответните точки от повърхнината на Лобел около съответните оси, за да получим желаният резултат на „прегъване със залепяне“.

#### Залепяне

Последната операция от метода на моделиране подобно на „Киригами“ е операцията залепяне. Чрез нея можем да добавим допълнителни триъгълници към съществуващата повърхнина. Това е важно, тъй като ни позволява да продължим с моделирането в дадена посока, когато триъгълниците в тази посока вече са се изчерпали и е необходимо удължаването на повърхнината. Друг случай, в който операцията „залепяне“ би могла да ни бъде полезна, е ако имаме съществуваща повърхнина на Лобел, в която поради някаква причина има дупки по средата на самата повърхнина. Такива дупки могат да се получат или от рязане на повърхнината, или от алгоритъм за генериране на такава повърхнина, който поради някаква причина е оставил липсващи триъгълници в средата. Такива липсващи триъгълници могат лесно да бъдат добавени с операцията „залепяне“.

Нека покажем нагледно какво се има предвид под „залепяне“, продължавайки моделирането на примера от предните три секции („рязане“, „прегъване“ и „прегъване със залепяне“). За целта ще добавим няколко реда с триъгълници към дясната половина на дотук измоделираната повърхнина на Лобел. Първо селектираме точката в долния десен ъгъл и точката в горния десен ъгъл на повърхнината. Тези две точки дефинират ръба, от който ще започне залепянето. След това селектираме трета точка, която ще дефинира полуравнината, в която ще добавяме редове с триъгълници. С въвеждане от клавиатурата можем да специфицираме колко точно редове искаме да добавим, като в примера по-долу сме избрали да добавим три реда с триъгълници. Натискайки Enter за потвърждение, операцията е завършена и повърхнината е удължена с допълнителните триъгълници. На долната картинка може да видите етапите, през които протича залепянето и крайният резултат.



## Моделиране с апроксимиращи алгоритми

Както видяхме в предишния метод, със средствата на Киригами бихме могли да измоделираме напрактика всяка една повърхнина изградена изцяло от равностранни триъгълници. Това обаче може да се окаже доста трудоемка и времеемка задача, изискваща особена съобразителност, за да може човек да избере правилните операции в правилния ред и да стигне до желания резултат. Затова нашето приложение ще предложи втори метод за моделиране, при който само с едно кликане на мишката ще можеш да създадеш комплексна повърхнина на Лобел, приближаваща някаква друга параметрично дефинирана повърхнина.

В тази секция ще покажем 4 различни разновидности на такива алгоритми за апроксимиране с повърхнина на Лобел. За по-лесно обясняване на логиката на алгоритмите преди това ще дефинираме няколко термина, които се използват от всичките 4 алгоритъма.

#### Дискретна UV мрежа

Нека имаме пространствена повърхнина, дефинирана с функция с два параметъра **u** и **v**, която е непрекъсната в интервала [0, 1] относно всеки от параметрите.



**U-линия** ще наричаме линия от повърхнината, за която стойността на u параметъра е константа. **V-линия** ще наричаме линия от повърхнината, за която стойността на v параметъра е константа. Нека сега от повърхнината **r** вземем нейните **u** и **v** линии, избирайки последователни стойности на параметрите u и v, така че интервалът [0, 1] да бъде разделен на равни подинтервали. Нека накрая, използвайки пресечните точки на **u** и **v** линиите, да добавим по два триъгълника във всеки от пространствените четириъгълници, получени между съседните линии. Така получената мрежа от триъгълници ще наречем **дискретна UV мрежа**.

Казаното по-горе може да опишем с малко по-конкретни формули. Нека „нарежем“ **r** функцията на **n** парчета в посока на **u** линиите и **m** парчета в посока на **v** линиите. Тогава за всяка от стойностите на параметрите **u** и **v** можем да кажем следното.



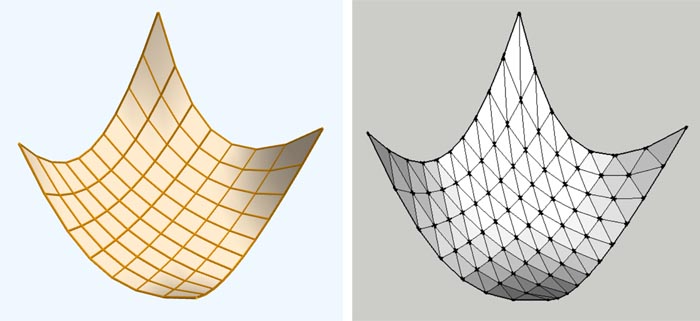
Пресечните точки на **u** и **v** линиите можем да запишем по следния начин.



Тогава дискретната UV мрежа се дефинира от всички триъгълници между съседните **u** и **v** линии дефинирани по следния начин.



На долната картинка е показана една примерна повърхнина **r**, върху която са нарисувани **u** и **v** линиите при стойности на **n** и **m** съответно 7 и 10. От дясната страна е показана съответващата дискретна UV мрежа при същото деление на повърхнината.



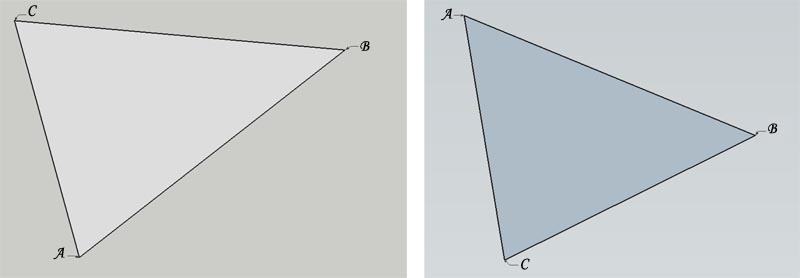
Алгоритмите, които ще опишем след малко, могат да приближават дискретна UV мрежа с повърхнина на Лобел. Както е видно, колкото повече деления изберем за оригиналната повърхнина **r**, толкова по-близо до нея ще бъде и съответната дискретна мрежа. Затова за целите на апроксимиране с повърхнини на Лобел можем да кажем, че алгоритмите, които ще покажем, са подходящо решение за апроксимиране на произволна параметрична повърхнина, която е непрекъсната в интервала на двата си параметъра [0, 1].

#### Окта-Тетра мрежа

Нека разгледаме следната рекурсивна структура, състояща се само от еднакви равностранни триъгълници. Като първа стъпка да изберем някъде в пространството един равностранен триъгълник **ABC**. Ще гледаме на този триъгълник като на ориентирана тройка от точки, като повърхността на триъгълника, виждаща се от едното полупространство, ще наричаме „лице“ на триъгълника и ще я оцветяваме в бяло на схемите, а повърхността, виждаща се от другото полупространство ще наричаме „гръб“ на триъгълника и ще я оцветяваме в синьо. Полупространството, от което ще виждаме лицето на ориентирания триъгълник ABC, ще бъде дефинирано от посоката на нормалния вектор **n** за триъгълника, който ще бъде пресмятан по следния начин.



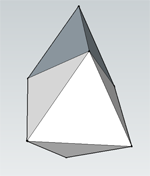
На долната картинка могат да се видят два перспективни погледа към началният ориентиран триъгълник **ABC** – единият, гледащ към лицето на триъгълника, а другият - гледащ към гърба му.



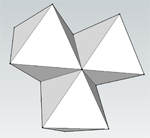
На следващата стъпка от нашата рекурсивна структура ще направим следното – откъм лицето на първоначалния триъгълник ще добавим още 3 триъгълника, така че общо четирите получени триъгилника да образуват тетраедър. Също така ще искаме всички новосъздадени триъгълници да са ориентирани така, че лицето им да сочи към вътрешността на тетраедъра. Като резултат се получава следния тетраедър.



На следващата стъпка нека добавим към първоначалния триъгълник още 7 триъгълника от страната на неговия гръб. Тези общо 8 триъгълника ще бъдат подредени така, че да се получи един октаедър. Освен това новите триъгълници ще бъдат така ориентирани, че гърбовете на всеки от тях да гледат към вътрешността на октаедъра. Като резултат от предните две стъпки се получават залепени тетраедър и октаедър, които могат да се видят на картинката по-долу.

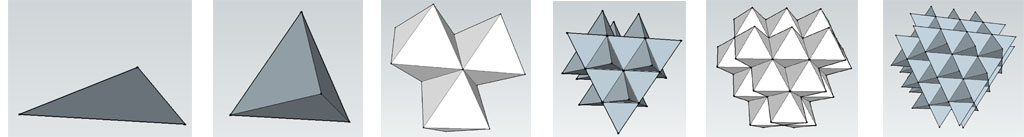


Тук е моментът да отбележим едно интересно наблюдение, която ще бъде използвано от нашата рекурсивна конструкция. Двустенните ъгли на октаедъра и тетраедъра се допълват до 180 градуса. Поради тази причина съседните странични триъгълници на октаедъра и тетраедъра са компланарни, както може да се забележи и на горната картинка. Сега нека на следващата стъпка на всеки от останалите 3 триъгълника от тетраедъра да добавим по един октаедър от страната на техните гърбове.



Както забелязваме, пространството между всеки два съседни октаедъра може да бъде запълнено от един тетраедър, за който само трябва да бъде добавен ръбът, свързващ срещуположните им свободни върхове.

Следвайки логиката от предните стъпки, можем да продължим рекурсията по следния начин. На една стъпка добавяме триъгълници, затварящи тетраедри. На следващата стъпка добавяме триъгълници, затварящи октаедри. След това пак тетраедри, после пак октаедри и т.н. Така можем да продължим нашата рекурсия до безкрайност, разширявайки получената мрежа във всички посоки. Това гарантира, че всяка точка от пространството в някакъв момент от рекурсията ще попадне или във вътрешността на тетраедър, или във вътрешността на октаедър, или на граничен триъгълник между съседни октаедър и тетраедър. За всеки триъгълник от нашата мрежа можем да кажем, че е равностранен, еднакъв на всички останали от мрежата, от страната на лицето му винаги започва един тетраедър, докато от страната на гърба му винаги започва един октаедър. На долната картинка могат да се видят последователните стъпки от рекурсията, като се започне от един ориентиран триъгълник и на всяка следваща се редува добавянето на нови тетраедри и нови октаедри към структурата.



Тази рекурсивна структура от еднакви равностранни ориентирани триъгълници ще наречем **Окта-Тетра мрежа**. Такава мрежа ще бъде използвана от всички наши следващи алгоритми, като на всяка стъпка от алгоритмите ще се избира подходящ триъгълник от мрежата така, че да се апроксимира най-добре дадена дискретна UV мрежа с повърхнина на Лобел.

#### Ориентирана проекционна площ

За целта на нашите алгоритми ще търсим близостта между триъгълниците на апроксимираната и апроксимиращата мрежа на базата на проектирани обеми, заключени между двете повърхности. За да обясним по-лесно идеята в тримерното пространство, обаче, ще започнем първо с обяснение на аналогична идея в равнинния случай, където ще можем да покажем чертежи и изчисления. В равнинния случай, за да търсим близост между отсечка и линия ще дефинираме понятието „**ориентирана проекционна площ**“.

Нека имаме една равнинна параметрична крива **r**, зададена параметрично с векторна функция на един параметър, както следва.



Нека също така имаме една отсечка с краища точките **A** и **B**. Дефинираме функция **p** върху тази отсечка, която ще дава разстоянието от всяка точка на отсечката до най-близката проектирана точка от линията **r**. Това разстояние ще е ориентирано, като ще го взимаме със знак плюс за точки от **r**, които са в едната полуравнина спрямо **AB** и със знак минус за точки от **r**, които са в другата полуравнина спрямо **AB**. За целите на задачата няма голямо значение коя полуравнина ще изберем с положителни стойности на **p(t)** и коя - с отрицателни. За функцията **p** можем да кажем следните неща.



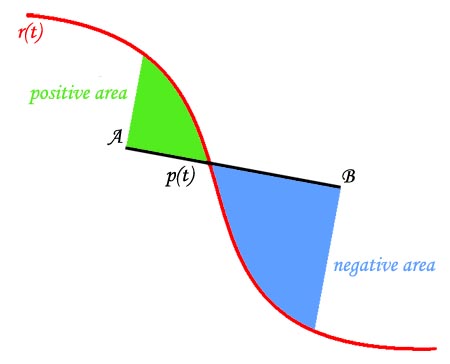
Възможно е за някои точки от отсечката **AB** да не съществува проекция на точка от **r**. За такива точки ще додефинираме функцията **p** и ще кажем, че за тях **p** има стойност 0.



Така, след като вече имаме дефинирана функцията **p** в целия интервал [0, 1], вече можем да кажем какво имаме предвид под **ориентирана проекционна площ** **SAB** на отсечката **AB**. А именно, тя e модул от сумираната ориентирана площ заключена между отсечката **AB** и линията **r**.



За по-голяма яснотa на описаните термини може да погледнете примерния четеж по-долу.



На чертежа положителната площ е обозначена със зелен цвят, а отрицателната - със син. Така или иначе, както вече обяснихме, за целта на задачата не е от значение коя от площите ще бъде смятана със знак плюс и коя - със знак минус, тъй като след като площите се сумират, резултатът за **SAB** се взима с модул.

#### Ориентиран проекционен обем

След като изяснихме понятието „ориентирана проекционна площ“ вече можем да обясним аналогичното в тримерното пространство понятие „**ориентиран проекционен обем**“.

Нека имаме една пространствена повърхнина **r**, дефинирана с два параметъра.



Нека също така имаме един ориентиран триъгълник **ABC**. Дефинираме функция **p** на два параметъра върху триъгълника **ABC**, където **p** ще дава разстоянието от всяка точка на триъгълника до най-близката проектирана точка от повърхнината **r**. Това разстояние ще е ориентирано, като ще го взимаме със знак плюс за точки от **r**, които са в едното полупространство спрямо **ABC** и със знак минус - за точки от **r**, които са в другото полупространство спрямо **ABC**. За целите на задачата няма голямо значение кое полупространство ще изберем с положителни стойности на **p(u, v)** и коe - с отрицателни. За функцията **p** можем да кажем следните неща.



Възможно е за някои точки от триъгълника **ABC** да не съществува проекция на точка от **r**. За такива точки ще додефинираме функцията **p** и ще кажем, че за тях **p** има стойност 0.



Така след като вече имаме дефинирана функцията **p** в цялата дефиниционна област на параметрите **u** и **v**, вече можем да кажем какво имаме предвид под **ориентиран проекционен обем** **VABC** на триъгълника **ABC**. А именно, той е модул от сумирания ориентиран обем, заключен между триъгълника **ABC** и повърхнината **r**.

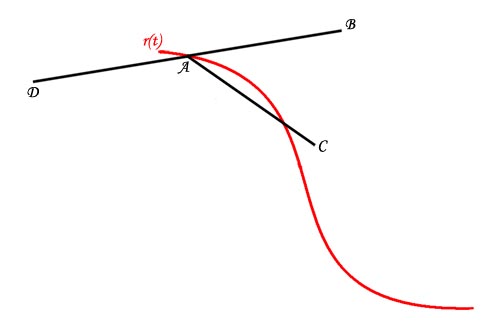


Както се вижда от формулата, сумираните обеми се взимат по модул и затова, както вече споменахме, е без значение в кое полупространство обемите ще ги смятаме със знак плюс и в кое - със знак минус.

#### Усреднено ориентирано проекционно разстояние

Нека сега изясним последното понятие, което ще ни е необходимо за описването на алгоритмите за апроксимиране. Отново първо ще покажем двумерния случай с подходящи примери и чертежи и впоследствие ще продължим към описанието в тримерното пространство, където дефинициите са аналогични.

Нека имаме следната поставена задача. Имаме равнинна линия, определена с векторното параметрично уравнение с един параметър **r(t)**. Нашата задача е да апроксимираме тази равнинна линия с равни по дължина отсечки, които се избират от дадено множество с отсечки. На долната картинка е показана линията **r** и три отсечки **AB**, **AC** и **AD**, от които трябва да изберем точно една. Целта е избраната отсечка да „апроксимира“ най-добре дадената крива.



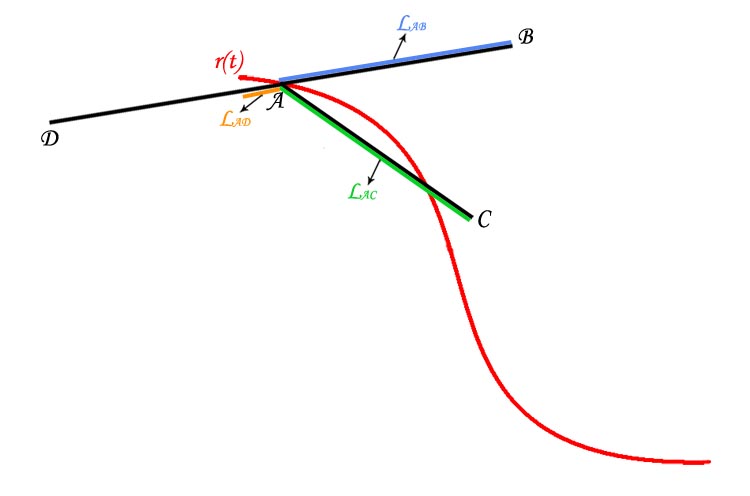
Чисто на око, поглеждайки задачата, изглежда, че отсечката **AC** е най-близо до желаната линия и вероятно тя е най-подходящия избор от даденото ни множество отсечки. Как обаче бихме могли да изчислим тази близост, така че да изберем правилната отсечка. Нека опитаме с вече дефинирания термин „ориентирана проекционна площ“ и да видим какъв резултат ще получим в случая. Нека първо сравним проекционните площи **SAB** и **SAC** съответно на отсечките **AB** и **AC**. Отсечката **AB** има проекционни разстояния по цялата си дължина и те, освен че са доста големи, са и само в едната полуравнина. Това прави **SAB**  доста голямо като стойност. От своя страна **AC** също има проекции по цялата си дължина, но те, освен че са доста по-малки по дължина, също така са и в двете полуравнини спрямо отсечката. По този начин отрицателните и положителните площи се унищожават частично и **SAC** има гарантирано по-малка стойност.



До тук изглежда, че ориентираният проекционен обем може да свърши добра работа за изчисляване на близост за отсечки, при които има проекции по цялата им дължина. Нека видим, обаче, какво можем да кажем за **SAD**. Отсечката **AD** в по-голямата си част изобщо няма проекции, поради, което в по-голямата си част нейната проекционна площ според дефиницията остава на минималната си нулева стойност. Единствено в малка част в единия край на отсечката има малка проекционна площ, която, обаче, въпреки че е едностранна, все пак е по-малка от сумарната площ на отсечката AC.



Оттук забелязваме, че като че ли ориентираната проекционна площ няма да ни свърши работа за гранични случаи, при които някоя отсечка има твърде голяма дължина без дефинирани проекционни разстояния. За решаването на този проблем ще въведем понятието „**усреднено ориентирано проекционно разстояние**“. На долната картинка може да видите означени с различни цветове дължините от отсечките, които имат проекции на линията **r** върху тях. Тези дължини ще кръстим с **LAB**, **LAC** и **LAD** и на долната картинка те могат да бъдат видeни съответно в синьо, зелено и оранжево.



**Усреднено ориентирано проекционно разстояние** на отсечката **AB** ще наричаме отношението между **SAB** и **LAB** и същото ще го означаваме с **DAB**.



Тъй като отсечките AB и AC са равни по дължина и имат проекции по цялата си дължина, то **LAB** и **LAC** са равни. Това от своя страна гарантира запазване на неравенството между проекционните им площи, като то ще бъде със същата посока при усреднените им проекционни разстояния.



Нека видим какво се случва със сравнението на близостта между **AC** и **AD**. Тъй като **AD** има в пъти по-малка проекционна дължина в сравнение с **AC**, то при делението на **LAD** стойността на **DAD** ще се увеличи значително. Това ще позволи да се обърне неравенството в полза на отсечката **AC**.



Така критерият „усреднено ориентирано проекционно разстояние“ изглежда по-подходящ при избор на апроксимиращи отсечки. На практика това разстояние е отношение между площ и дължина, което го прави с мерна единица за дължина. На практика то ни дава средното разстояние между отсечката и линията **r**, откъдето идва и името му.

Сега, след като изяснихме това понятие за двумерния случай, нека прехвърлим разсъжденията към аналогичния тримерен случай. В следващите алгоритми за апроксимиране ние ще разполагаме на всяка стъпка с някакво множество от еднакви равностранни триъгълници и ще трябва да изберем най-добрия триъгълник, който апроксимира най-добре дадена тримерна повърхнина **r**. Аналогично на двумерния случай, ще забележим, че в граничните случаи, при които голяма част от даден триъгълник **АBC** няма проекции върху себе си, ориентираният проекционнен обем **VABC** може да се окаже не съвсем подходящо средство за сравняване. Затова ще пресмятаме площта от триъгълника **ABC**, върху която реално има проекции на повърхнината **r**. Тази проекционна площ ще означим с **SABC**. По този начин в тримерния случай отново ще дефинираме понятието „**усреднено ориентирано проекционно разстояние**“ **DABC** като отношение на **VABC** и **SABC**.



Във тримерния случай **DABC** е отношение на обем и площ и мерната му единица отново е дължина. В крайна сметка, както и в двумерния случай, тази стойност представлява средното проекционно разстояние между точките в триъгълника и точките от повърхнината, което обяснява и името „усреднено ориентирано проекционно разстояние“.

Накрая ще споменем накратко случая, в който **SABC** е нула, обяснявайки защо този случай не трябва да ни притеснява. Когато тази площ е нула, следва, че триъгълника **ABC** няма никакви проекции върху себе си или ако има, то те образуват множество с нулева площ (например отсечка или точка). Такива триъгълници можем да кажем, че са неподходящи за апроксимиране на желаната повърхнина **r** и тях можем да ги изключим от сравненията и сметките, касаещи **DABC**.

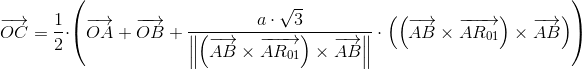
#### Апроксимиращи алгоритми – обща информация и начални условия

Следващите четири секции ще разяснят действието на четири различни вариации на апроксимиращи алгоритми. Алгоритмите са подредени хронологично според тяхното появяване, като всеки следващ се опитва да подобри резултата от апроксимацията на предходния.

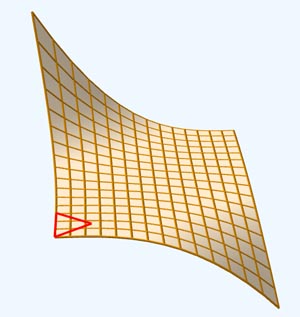
Всичките алгоритми, които ще предложим, са базирани на една Окта-Тетра мрежа, от която се избират апроксимиращи триъгълници на всяка стъпка от итерацията. Тук е моментът да кажем, че този принцип на изчисления на алгоритмите ограничава получените резултати до подмножество на всички повърхнини на Лобел. Това подмножество ползва точно две стойности на двустенните ъгли между съседните триъгълници – двустенен ъгъл на октаедър и двустенен ъгъл на тетраедър. Реално повърхнините на Лобел биха могли да имат и двустенен ъгъл на икосаедър, както и други произволни ъгли в части от повърхнината, където тя е само с единична кривина.

Като входни данни алгоритмите изискват точно два параметъра, за да започнат апроксимацията. Първият параметър е повърхнината, която трябва да бъде апроксимирана. Алгоритмите в това приложение работят с дискретни UV мрежи като входен параметър. Вторият параметър е размерът на страната на равностранните триъгълници в резултатната повърхнина на Лобел. Знаейки апроксимираната дискретна UV мрежа и размера на равностранния триъгълник, алгоритъмът започва пресмятането на апроксимираща повърхнина на Лобел.

Всеки от алгоритмите започва пресмятането с поставянето на първия апроксимиращ равностранен триъгълник. Първият триъгълник се слага в единия ъгъл на UV мрежата, като идеята при неговото позициониране е той да се допира до повърхнината в този й край. Нека този първи триъгълник бъде кръстен с **ABC**. Нека също така страната на равностранния триъгълник да бъде с размер **а**. За точките от апроксимираната дискретна UV мрежа ще използваме означението **Rij** от [секцията за дискретни UV мрежи](#_Дискретна_UV_мрежа). С тези означения можем да пресметнем точките **A**, **B** и **C** по следния начин.



На долната картинка може да се види примерна апроксимирана повърхнина с нейните UV линии и първият апроксимиращ равностранен триъгълник, който е нарисуван с червено.



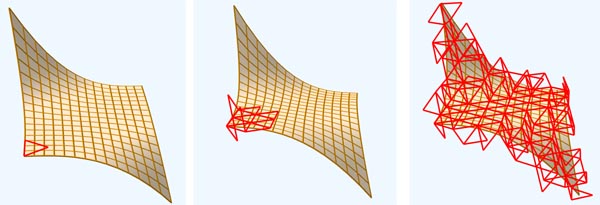
Поставяйки първия ориентиран триъгълник, алгоритъмът еднозначно определя Окта-Тетра мрежата, от която впоследствие ще избира следващите триъгълници. Както беше споменато в [секцията за Окта-Тетра мрежите](#_Окта-Тетра_мрежа), характерно за тази геометрична структура е, че до всеки триъгълник има тетраедър от лицевата му страна и октаедър от страната на неговия гръб.

За работата си всеки от алгоритмите използва една опашка от рекурсивни стъпки. Всяка рекурсивна стъпка се добавя в опашката и когато дойде нейният ред в рекурсията, алгоритъмът я обработва. Всяка рекурсивна стъпка носи информация за множество равностранни триъгълници, от което множество алгоритъмът би могъл да избере един, счетен за най-близък до повърхнината. Критерият за избора на най-добрия триъгълник е [усредненото ориентирано проекционно разстояние](#_Усреднено_ориентирано_проекционно) между триъгълника и апроксимирантата повърхнина. Триъгълникът с най-малко такова разстояние се приема за най-близък до повърхнината и става част от финалната апроксимираща повърхнина на Лобел. Сметките приключват, когато рекурсивните стъпки в опашката свършат или когато някое специфично за алгоритъма условие е изпълнено.

Ако трябва да опишем действието на алгоритъма с точки, то те биха изглеждали по следния начин:

1. Избира се първият равностранен триъгълник.
2. Към опашката на рекурсивните стъпки се добавят първите стъпки, съдържащи триъгълници, съседи на първия.
3. Изважда се рекурсивна стъпка от опашката.
4. Избира се най-близък до повърхнината триъгълник от предложеното множество в текущата рекурсивна стъпка. В случай, че никой от триъгълниците няма проекционна площ, тогава не се избира нито един триъгълник и се отива направо към **точка 6**.
5. Към опашката на рекурсивните стъпки евентуално се добавят нови стъпки, съдържащи триъгълници, съседи на последния избран.
6. Проверява се дали алгоритъмът трябва да приключи. Краят настъпва, ако са свършили стъпките в рекурсивната опашка или е изпълнено някакво специфично за алгоритъма условие. Ако не е настъпил краят, тогава се връщаме към **точка 3**.

На долната картинка могат да се видят 3 кадъра от действието на един примерен алгоритъм - изборът на първия триъгълник, резултатът от изборите в първите няколко рекурсивни стъпки и резултатните триъгълници към края на сметките на алгоритъма.

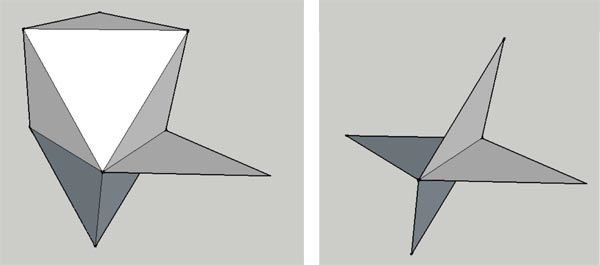


Разликите в различните алгоритми ще са основно в точка 5, където трябва да се създадат новите рекурсивни стъпки на база съседите на последния избран триъгълник. Нека сега разгледаме по-детайлно всеки алгоритмите по отделно с примери.

#### Lobel Mesh Projecting Algorithm

Името на алгоритъма идва от следната специфика – при избора на най-добър триъгълник в **точка 4** от изпълнението, алгоритъмът проектира точки от апроксимираната дискретна UV мрежа в равнината на триъгълника. Точките **Rij**, които се проектират във вътреността на триъгълника се маркират като „покрити“. Алгоритъмът приключва действието си в **точка 6**, ако всички **Rij** бъдат маркирани като покрити. За по-кратко ще назоваваме алгоритъма с абриевиатурата му **LMP** (Lobel Mesh Projecting).

Сега да изясним как **LMP** избира следващите рекурсивни стъпки, за да ги добави в опашката на рекурсията в **точка 5**. За всеки от трите ръба на триъгълника се проверява дали може да се създаде рекурсивна стъпка. Такава рекурсивна стъпка ще съдържа съседните на ръба триъгълници, от които в последствие трябва да се избере точно един, който има най-малко усреднено ориентирано проекционно разстояние до апроксимираната мрежа. Както може да се види на долната картинка, до всеки ръб на триъгълник има точно по един съседен октаедър и един съседен тетраедър. На втората част от картинката могат да се видят само триъгълниците от октаедъра и тетраедъра, които съдържат съответния ръб.



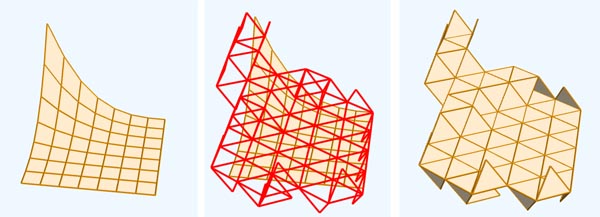
От картинката е видно, че всяка следваща рекурсивна стъпка ще съдържа точно 3 триъгълника, съседни на предния избран. Сега остава да разберем критерият, по който **LMP** решава дали да съзаде рекурсивната стъпка или не. Рекурсивната стъпка се създава само ако никой от трите съседни триъгълника не е бил създаван от предходна рекурсивна стъпка на алгоритъма. Логиката, която се крие зад този критерии е, че ние целим да създадем повърхнина на Лобел и не искаме да имаме повече от два триъгълника с общ ръб.

Нека видим как биха изглеждали точките, описващи работата на **LMP**.

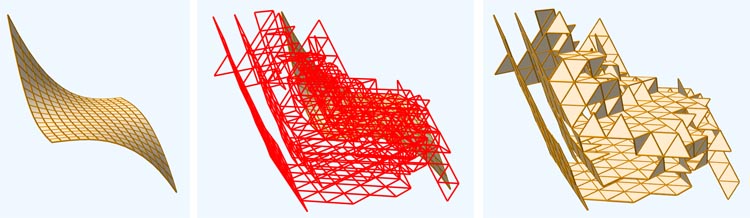
1. Избира се първият равностранен триъгълник.
2. Към опашката на рекурсивните стъпки се добавят първите три рекурсивни стъпки за всеки от ръбовете на триъгълника.
3. Изважда се рекурсивна стъпка от опашката.
4. Избира се най-близък до повърхнината триъгълник от предложеното множество в текущата рекурсивна стъпка. В случай, че никой от триъгълниците няма проекционна площ, тогава не се избира нито един триъгълник и се отива направо към **точка 6**. Освен това точките от UV мрежата се проектират върху избрания триъгълник и тези, които попаднат във вътрешността му, се маркират като „покрити“.
5. Към опашката на рекурсивните стъпки евентуално се добавят нови стъпки, съдържащи триъгълници, съседи на последния избран. За всеки от ръбовете се добавя по една рекурсивна стъпка, в случай че съответната стъпка няма да съдържа вече използван от алгоритъма триъгълник.
6. Проверява се дали алгоритъмът трябва да приключи. Краят настъпва, ако са свършили стъпките в рекурсивната опашка или ако всички точки **Rij** от UV мрежата вече са маркирани като „покрити“. Ако не е настъпил краят, тогава се връщаме към **точка 3**.

Да видим няколко примера за практическата работа на алгоритъма.

В първият пример ще изберем една проста повърхнина за приближение, която има огъване само в единият от четирите ъгъла от контрура й. Ще приближим тази повърхнина с приблизително големи равностранни триъгълници, така че ще е нормално да очакваме „назъбен“ резултат от апроксимацията. На долните три картинки се виждат първоначалната UV мрежа, приближението на LMP върху UV мрежата и приближения резултат, показан без UV мрежата. Това, което можем да забележим е, че като че ли алгоритъмът се е справил с основната задача да последва единствената извивка на приближаваната повърхнина.



Нека изпробваме един малко по-сложен случай. В него приближаваната повърхнина ще има извивки в два срещуположни ъгъла. На долната картинка може да се види резултатът от апроксимацията.



Тук, обаче, виждаме нещо доста притеснително. LMP вместо да се движи до повърхнината, всъщност е направил нещо като трислойно приближение, което е далеч от желания от нас резултат.

Защо се случва този проблем при LMP? Обяснението е доста просто и може да бъде видяно, поглеждайки точките по работата на алгоритъма. Критерият за близост, който е в точка 4, работи само локално. Избира се най-близкият триъгълник от дадени 3 възможни. За тези 3, обаче, по никакъв начин не се гарантира, че ще са достатъчно близко до повърхнината поради липсата на глобален критерий за близост. Реално рекурсията се движи във всички посоки и единственото, което я спира, е допълнителния критерий за „покриване“ на точките **Rij**, чрез проектирането им. Докато все още има непокрити точки, LMP продължава действието си в нежелана грешна посока, което резултира в многослойност на получения резултат. Следващият алгоритъм, който ще представим, ще опита да реши проблема за глобална близост на LMP.

#### Centroids Distance Measuring Algorithm

Името на този алгоритъм идва от факта, че в него ще въведем критерии за локална близост като измерваме разстоянията между центровете на октаедрите и тетраедрите от Окта-Тетра мрежата и ще използваме само тези от тях, които са на определено разстояние от приближаваната UV мрежа. За краткост ще назоваваме този алгоритъм с абривиатурата му **CDM** (Centroids Distance Measuring).

**CDM** се различава от **LMP** най-вече в стъпки 2 и 5, когато става дума за създаването на новите рекурсивни стъпки. CDM отново се опитва за всеки ръб на последно избрания триъгълник да добави рекурсивна стъпка със съседите на ръба. Разликата е, че CDM взима центровете на съседните октаедър и тетраедър и измерва разстоянието от тези центрове до повърхнината. Това разстояние е обикновено евклидово разстояние и се има предвид най-късата отсечка, свързваща центъра на многостена с точка от дискретната UV мрежа, която приближаваме. На практика сметките се свеждат до намиране на разстояние между точка и триъгълник в тримерното пространство. Тази задача, от своя страна, се свежда до проверка с няколко възможни гранични състояния. Ако точката се проектира във вътрешността на триъгълника от UV мрежата, то разстоянието е точно дължината на проекционната линия. Ако обаче точката се проектира извън триъгълника, тогава можем да решим първо равнинната задача за намирането на разстоянието между проекцията в равнината на триъгълника и самият триъгълник. Разстоянието от центъра на многостена до триъгълника тогава може да бъде сметнато с Питагорова теорема, знаейки височината на проекцията в равнината на триъгълника и разстоянието между проекцията в равнината и триъгълника.

Какъв обаче трябва да е критерият за това разстояние и кои многостени трябва да избираме? Ключът за това решение се крие в непрекъснатостта на апроксимираната UV мрежа и във факта, че Окта-Тетра мрежата, от която ще избираме равностранните триъгълници, запълва цялото пространство, без да оставя празнини между октаедрите и тетраедрите. Тези два факта гарантират следното твърдение – ако изберем множеството от всички октаедри и тетраедри, които пресичат UV мрежата, то това множество се състои от свързани многостени и тези многостени съдържат в обема си всички точки от апроксимираната мрежа. Знаейки това, можем да си поставим за цел да спазваме глобален критерии за близост, който ще ни позволи да изберем многостени от Окта-Тетра мрежата, които са достатъчно близко, за да могат да пресичат апроксимираната мрежа. Такъв критерии ще е разстоянието от центъра на многостена до повърхнината да е по-малък от радиуса на описаната около многостена сфера. Ако разстоянието е по-голямо от радиуса, тогава със сигурност няма как да има пресичане. Ако е по-малко – има шанс да има пресичане. Макар и да рискуваме с този критерии да селектираме и някои многостени, които не пресичат мрежата, все пак те ще са достатъчно близко, за да можем да кажем че „почти“ я пресичат и също така да ги изберем като подходящи.

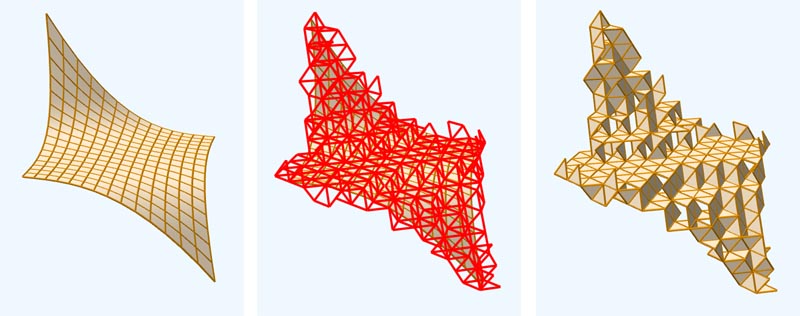
В крайна сметка, знаейки този критерии за близост, вече можем да опишем точките за изпълнение на **CDM**.

1. Избира се първият равностранен триъгълник.
2. Към опашката на рекурсивните стъпки се добавят първите три рекурсивни стъпки за всеки от ръбовете на триъгълника, като се избират триъгълници само от многостени, чиито центрове са по-близо до повърхността от радиуса на описаната им сфера.
3. Изважда се рекурсивна стъпка от опашката.
4. Избира се най-близък до повърхнината триъгълник от предложеното множество в текущата рекурсивна стъпка. В случай, че никой от триъгълниците няма проекционна площ, тогава не се избира нито един триъгълник и се отива направо към **точка 6**.
5. Към опашката на рекурсивните стъпки евентуално се добавят нови стъпки, съдържащи триъгълници, съседи на последния избран. За всеки от ръбовете се добавя по една рекурсивна стъпка, като се избират триъгълници само от многостени, чиито центрове са по-близо до повърхността от радиуса на описаната им сфера. Освен това не се избират триъгълници, които вече са били избирани за някоя предишна рекурсивна стъпка.
6. Проверява се дали алгоритъмът трябва да приключи. Краят настъпва, ако са свършили стъпките в рекурсивната опашка. Ако не е настъпил краят, тогава се връщаме към **точка 3**.

Както виждаме, **CDM**, за разлика от **LMP**, не използва критерии за „покриване“ на точки от UV мрежата и единственият критерий за край на алгоритъма е, когато се изпразни опашката с рекурсивните стъпки. Сигурни сме, че алгоритъмът ще завърши успешно, тъй като многостените, които отговарят на глобалния критерии за близост, са краен брой.

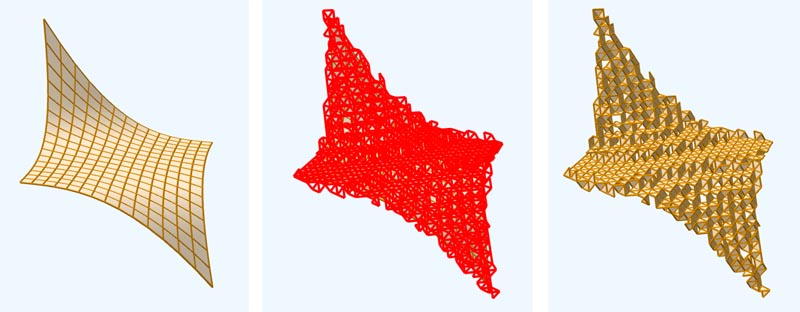
Нека сега разгледаме някои примери за работата на алгоритъма.

В първия пример ще вземем отново повърхнината с огъване в двата ъгъла. Гледайки долната картинка с резултатите от **CDM** върху тази повърхнина, можем да кажем, че критерият за глобална близост определено е свършил работата си и е подобрил значително апроксимацията на **CDM** спрямо **LMP**. Единствените проблеми, които забелязваме, са в някои локални зони от повърхнината на Лобел, където има стърчащи навън от повърхнината триъгълници и точно до тях има дупки.



Сякаш част от триъгълничетата са се „отворили“ в смисъла на „прозорец с панти“, и това е предизвикало дупки в резултатната апроксимация. Този проблем би могъл лесно да се реши, използвайки метода за моделиране със средствата на Киригами. Било то с огъване на „отворилите се“ триъгълничета, или с изрязването им и запълване на дупките с операцията „залепи“, ползвателят на приложението ще може да оправи локалните проблеми и да използва резултатът от **CDM** успешно. Това би било доста неприятно обаче, ако тези локални проблеми се случват твърде често за по-сложни мрежи. Нека видим няколко примера с такива мрежи.

Следващият пример е с приближение на същата UV мрежа, но този път с по-малка страна на равностранния триъгълник. Както може да се види на долната картинка, резултатът от апроксимацията на **CDM** изглежда още по-добър заради по-малката страна. Макар и все още да има локални проблеми с дупки, то те са пренебрежимо рядко явление.



Нека последно пробваме един пример на по-сложна UV мрежа, която е получена от повърхнина с двойна кривина. На долната картинка е показана една UV мрежа, наподобяваща стол и нейното приближение с повърхнина на Лобел получено със **CDM**.



Резултатът е впечатляващо близък до желания и проблемите с дупките са почти незабележими.

И все пак, въпреки че CDM работи добре, дали бихме могли да оправим дребните проблеми, забелязани в апроксимацията. Нека първо оправим единия от потенциалните проблеми, за който вече споменахме по-горе в описанието на алгоритъма. А именно, нашият критерий за близост цели избирането само на пресичащи се октаедри и тетраедни, но на практика с поставеното условие за разстояние, по-малко от радиуса на описаната сфера, ние избираме също така и някои непресичащи мрежата многостени. Да видим дали решението на този проблем би намалило проблемите в апроксимацията. Това ще разберем в следващия алгоритъм.

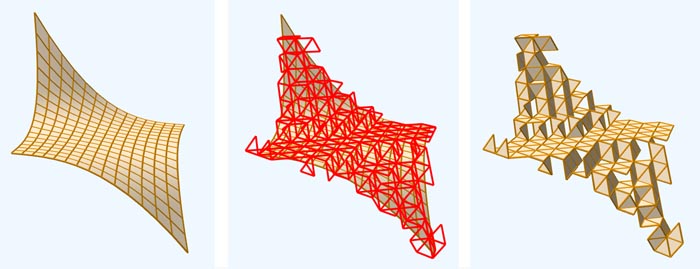
#### Intersecting Volumes Finding Algorithm

Името на този алгоритъм идва от факта, че този път критерият за близост ще бъде не разстояние, а условие за пресичане на обема на октаедъра или тетраедъра с приближаваната UV мрежа. Пресичането на всеки от многостените с дискретната UV мрежа ще става за всеки триъгълник. Иначе казано, всеки от триъгълниците на октаедъра или тетраедъра ще бъде пресичан с всеки от UV мрежата, докато не установим наличието на пресичане. Ако такова пресичане липсва, тогава съответният октаедър или тетраедър ще се счита за негоден за апроксимацията и няма да избираме триъгълници от него за резултатната повърхнина на Лобел. За краткост този алгоритъм ще назоваваме с абривиатурата **IVF** (Intersecting Volumes Finding).

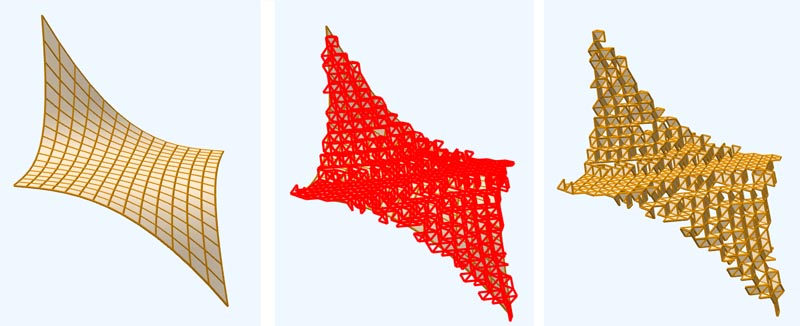
Нека сега разгледаме по точки работата на **IVF**. Единствената разлика със **CDM**, ще е в условието за създаване на нови рекурсивни стъпки в точки 2 и 5.

1. Избира се първият равностранен триъгълник.
2. Към опашката на рекурсивните стъпки се добавят първите три рекурсивни стъпки за всеки от ръбовете на триъгълника, като се избират триъгълници само от многостени, които пресичат апроксимираната UV мрежа.
3. Изважда се рекурсивна стъпка от опашката.
4. Избира се най-близък до повърхнината триъгълник от предложеното множество в текущата рекурсивна стъпка. В случай, че никой от триъгълниците няма проекционна площ, тогава не се избира нито един триъгълник и се отива направо към **точка 6**.
5. Към опашката на рекурсивните стъпки евентуално се добавят нови стъпки, съдържащи триъгълници, съседи на последния избран. За всеки от ръбовете се добавя по една рекурсивна стъпка, като се избират триъгълници само от многостени, които пресичат апроксимираната UV мрежа. Освен това не се избират триъгълници, които вече са били избирани за някоя предна рекурсивна стъпка.
6. Проверява се дали алгоритъмът трябва да приключи. Краят настъпва ако са свършили стъпките в рекурсивната опашка. Ако не е настъпил краят, тогава се връщаме към **точка 3**.

Нека видим с няколко примера какви промени ни носи различният глобален критерий за близост на многостените. Първият пример е с по-голяма страна на равностранния триъгълник.



Вторият пример е с по-малка страна на равностранния триъгълник, което естествено води и до по-добро приближение.



Забелязваме, че и в двата примера резултатите са близки до тези при **CDM**. Все още съществува и проблемът със стърчащите триъгълници и дупките до тях в мрежата на Лобел. Ако трябва да сме честни, дори бихме могли да кажем, че **IVF** работи по-зле от **CDM**, тъй като му отнема повече време докато пресметне финалния резултат. Това време се обяснява със значително по-бавните сметки за намиране на пресичане между триъгълници, сравнени със сметката за намиране на разстояние от точка до повърхнина. И все пак ще използваме IVF, като стъпка за постигането на следващия алгоритъм, който ще реши проблема с дупките в резултатната повърхнина на Лобел.

#### Intersecting Volumes Connecting Algorithm

Името на този алгоритъм идва от различният му подход за работа, а именно – на всяка стъпка ще бъде избиран точно по един триъгълник от пресичащ UV мрежата октаедър или тетраедър, в последствие на което между съседните многостени ще бъдат построени връзки, запълващи дупките в повърхнината на Лобел. За краткост ще назоваваме този алгоритъм с абривиатурата му **IVC** (Intersecting Volumes Connecting).

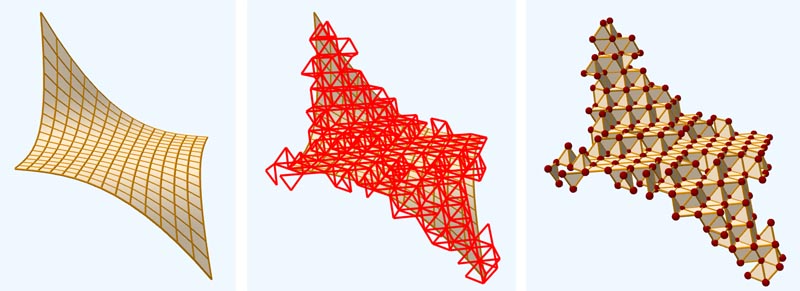
Нека първо изясним откъде идват проблемните дупки в мрежата при алгоритмите **CDM** и **IVF**, за да можем по-добре да обясним развитието при имплементацията на **IVC**. Макар и да използват глобален критерий за близост на триъгълниците, алгоритмите **CDM** и **IVC** продължават да използват локалния критерий за избор на точно един следващ съседен триъгълник измежду трите съседни за всеки ръб. Само че рекурсията се движи във всички посоки и нищо не ни гарантира, че избирайки триъгълник, който локално е по-близък, то няма по-добър триъгълник, който ще дойде от друга посока на рекурсията. От друга страна, ако по-подходящият триъгълник се появи на по-късен етап от рекурсията, то той няма да бъде избран тъй като преди него е избран първият, започващ от същия ръб. Както и преди бяхме споменали, тъй като резултатът трябва да е повърхнина, се прави проверка да не се получават по 3 триъгълника с обща страна. Всички тези разсъждения обясняват местата, на които някои от триъгълниците изглеждат „стърчащи“ от повърхнината, а триъгълника до тях, който изглежда по-подходящ, се оказва липсващ.

За да реши описаните проблеми, новият алгоритъм **IVC** ще създаде глобален критерий за близост, който ще определя еднозначно дали един триъгълник е подходящ за нашата мрежа или не. Този критерии ще разделя триъгълниците от нашата мрежа на два вида – такива, които са най-добри за своите октаедри или тетраедри и такива, които свързват съседните най-добри триъгълници, за да запълнят дупките в повърхнината на Лобел.

Ето как ще изглежда IVC по стъпки:

1. Избира се първият равностранен триъгълник.
2. Към опашката на рекурсивните стъпки се добавят първите рекурсивни стъпки за всеки от съседните многостени от Окта-Тетра мрежата. Избират се триъгълници само от многостени, които пресичат апроксимираната UV мрежа.
3. Изважда се рекурсивна стъпка от опашка.
4. Избира се най-близък до повърхнината триъгълник от предложеното множество в текущата рекурсивна стъпка. Ако рекурсивната стъпка не съдържа множество от свързващи триъгълници, тогава това е най-добрият триъгълник за даден многостен от Окта-Тетра мрежата. В случай, че никой от триъгълниците в стъпката няма проекционна площ, тогава не се избира нито един триъгълник и се отива направо към **точка 6**.
5. Към опашката на рекурсивните стъпки евентуално се добавят нови стъпки. За всеки от съседните многостени от Окта-Тетра мрежата, който пресича апроксимираната UV мрежа, се създават рекурсивни стъпки. Ако този съседен многостен вече е избрал своя най-добър триъгълник, то рекурсивните стъпки за този многостен ще съдържат свързващите триъгълници между двата многостена. В противен случай, ако този съседен многостен не е създавал рекурсивна стъпка за намиране на най-добър триъгълник, то такава стъпка се създава.
6. Проверява се дали алгоритъмът трябва да приключи. Краят настъпва, ако са свършили стъпките в рекурсивната опашка. Ако не е настъпил краят, тогава се връщаме към **точка 3**.

Нека покажем с пример как работят горните стъпки на **IVC**. На долните картинки се вижда приближението на повърхнината, която при CDM и IVF беше приближавана с дупки в мрежата. Както можем да забележим, обаче, IVC успешно запълва дупките, благодарение на вторичните рекурсивни стъпки от точка 5, които съдържат свързващи триъгълници.



В крайна сметка този алгоритъм дава най-завършен апроксимиращ резултат. Негов недостатък, заради многото сметки с пресичания на триъгълници и заради двустепенно итериране със свързващи триъгълници, е значително по-бавните изчисления. Ако CDM му отнема 2 секунди за построяване на апроксимацията, IVC би могъл да се забави до към 15 секунди за същата мрежа и същия размер на равностранните триъгълници. За това ако се търси по-бърза апроксимация на някоя по-голяма дискретна UV мрежа, е възможно да се избере CDM като достатъчно точен алгоритъм и малките проблеми в последствие да се дооправят с операциите за моделиране по метода на Киригами или като полученият резултат се дообработи във външна 3D програма, поддържаща OBJ файлов формат.

# Имплементационни детайли

В тази секция ще споменем по-интересните детайли от практическата имплементация на софтуера. Ще кажем по няколко думи за използваната технология, структури от данни, поддържани функционалности и други. Тъй като целият софтуер е десетки хиляди редове, тук ще споменем само най-важното. Пълната имплементация може да бъде видяна в моят Github акаунт: <https://github.com/deyan-yosifov/Deyan-Projects/tree/master/CSharp/FMI/LobelFrames> .

## Използвана технология

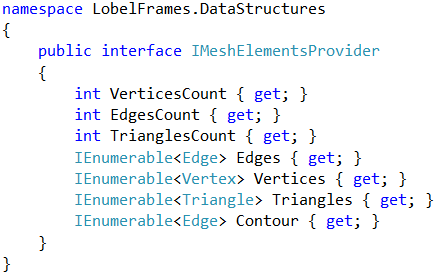
Софтуерът е написан на C#, използвайки [WPF (Windows Presentation Foundation)](https://en.wikipedia.org/wiki/Windows_Presentation_Foundation) технологията. Тъй като WPF е технологията за разработка на десктоп приложения за Windows, приложението ще изисква наличието на компютър с Windows и .Net Framework, за да се използва. За 3D визуализацията в самото приложение също е използвана единствено WPF технологията, така че не са необходими никакви странички тулове за подкарването му.

## Структури от данни

Тук ще споменем някои от най-важните структури от данни, необходими ни за работа с желаните 3D повърхнини.

#### IMeshElementsProvider

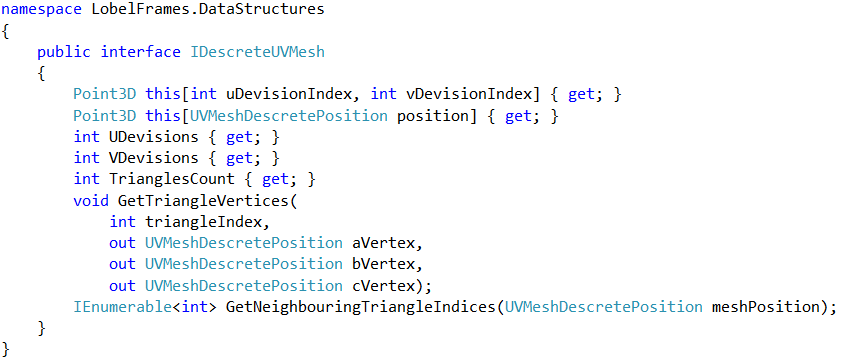
За да можем да работим по сходен и удобен начин с всички видове повърхнини в приложението, е използван интерфейс, наименован **IMeshElementsProvider**. Неговата задача е да уеднакви работата, що се отнася до основните елементи в представянето на една повърхнина, а именно – върхове, ръбове и триъгълници. Ето и как изглежда желаният интерфейс.



Тук е важно да се спомене, че имплементацията на класа **Triangle** пази в себе си инстанции на **Edge** и на **Vertex** съответно за своите страни и върхове. Важно условие, за да можем да моделираме повърхнините на Лобел (най-вече с метода на Киригами), е всеки два съседни триъгълника в мрежата да споделят еднакви инстанции за общите им върхове и ръбове. Това позволява операции като итериране по съседни елементи в мрежата, селектиране на съседни триъгълници (за да могат да бъдат прегънати, изрязани или залепени) и други. Затова всяка от операциите при моделиране на Киригами се грижи за запазване на връзките в мрежата. Алгоритмите за генериране на повърхнини на Лобел също се грижат да моделират повърхнини, които в последствие да могат да бъдат модифицирани със средствата на Киригами.

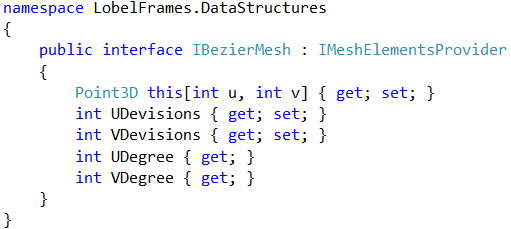
#### IDescreteUVMesh

За работата с дискретни UV мрежи е използван интерфейса **IDesceteUVMesh**. Той предоставя възможност за лесно боравене с точките и триъгълниците от мрежата. Боравенето с елементите от мрежата става посредством целочислени индекси. Ето и имплементацията на интерфейса.



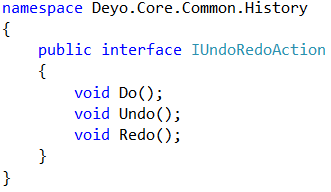
#### IBezierMesh

За да може приложението да предложи лесен и бърз начин за моделиране на примерни UV мрежи, е добавена възможност за моделиране на [повърхнини на Безие](https://en.wikipedia.org/wiki/B%C3%A9zier_surface). Тези повърхини са характерни с това, че с преместването на малък брой контролни точки, можем да получим една гладка и добре деформирана повърхнина. Освен това тази повърхнина е дефинирана в интервала [0, 1] за всеки от своите два параметъра и от нея лесно можем да извадим примерна дискретна UV мрежа, необходима ни за апроксимиращите алгоритми. Съответно имплементацията на интерфейса предоставя метод, който създава инстанция на **IDescreteUVMesh**.



## Команди

Командите в приложението са направени така, че всяка от тях да изпълнява някаква серия от действия, които могат да се отменят или правят отново в последствие, чрез Undo-Redo операции. Това е изключително важна функционалност, защото в противен случай най-малката грешка, която направи ползвателя на приложението по време на моделирането, щеше да налага започването на работата от начало. За да се имплементира тази функционалност, всяко действие, което има необходимост от връщане назад, имплементира интерфейса **IUndoRedoAction**.



## Файлови формати

За да има смисъл от моделирането в приложението, е важно да може резултатът да се запише и да се използва в последствие било то в същото приложение, или в друга програма, занимаваща се с моделиране на 3D обекти. Освен това е добре като първоначален вход на данните, ползвателят на приложението да може да зареди някакъв 3D обект, създаден в друго 3D приложение. По този начин би могъл да моделира повърхнините на Лобел, съпоставяйки ги референтно с някакъв друг желан обект. Например ако имаме някаква сграда и искаме да измоделираме покривното й остъкляване с повърхнини на Лобел, би било удобно да можем да вкараме контура на сградата в нашето приложение и да добавим към него необходимите триъгълници от мрежата на Лобел. За да могат да се поддържат всички тези сценарии, приложението имплементира записване и зареждане от два файлови формата.

#### OBJ файлов формат

Това е може би най-разпространеният формат за работа с 3D съдържание, който на практика се поддържа от всички 3D приложения. Нашето приложение също поддържа записване и зареждане на **OBJ** файл.

При записването се запазват триъгълниците на всички обекти в сцената. Това позволява моделираната в нашето приложение повърхнина на Лобел да бъде дообработена във външно приложение.

При отваряне на **OBJ** файл в приложението се зареждат всички 3D обекти като немодифицируеми. Тъй като форматът не позволява запазване на специфични повърхнини като Безие и Лобел, нашето приложение няма как да разбере какъв е вида на вкараната повърхнина и затова тя не може да се модифицира, а само се визуализира. Единствените позволени операции над такива повърхнини е тяхното селектиране, преместване или изтриване.

#### LOBZ файлов формат

Това е собственият файлов формат, който само нашето приложение поддържа. Той е полезен в случаите, в които искате да запишете текущата сцена и да продължите работата по нея по-късно. Този файлов формат пази информация за вида на всяка от повърхнините в сцената (Безие, Лобел или немодифицируема повърхнина). Също така се пази текущият поглед на камерата и текущата селекция в приложението (в случай, че е селектирана някоя от повърхнините). **LOBZ** форматът е бинарен формат и той компресира данните така, че заема в пъти по-малко място от OBJ файлът.

# Помощ за приложението