**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: Смартфон приложение за решаване на стереометрични задачи във виртуална реалност

Дипломант: Научен ръководител:

*Владимир Владинов Александър Ангелов*

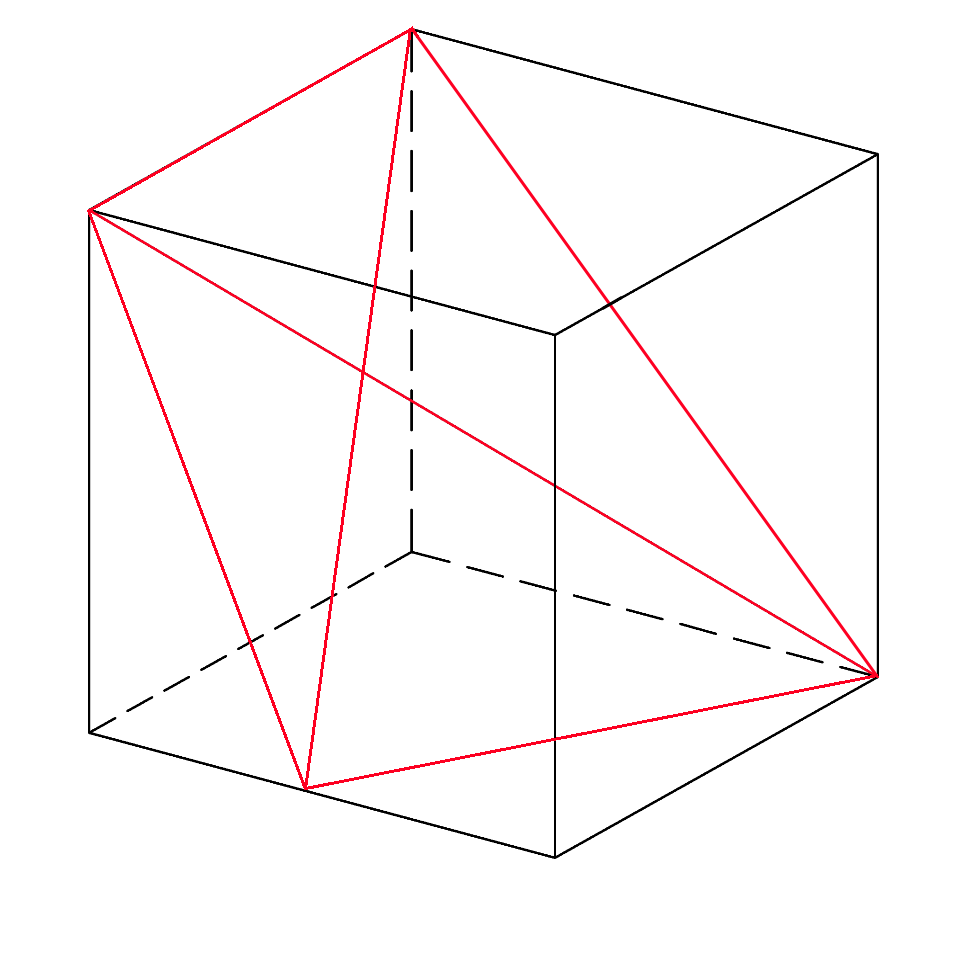
СОФИЯ

2019

**УВОД**

“Axyno” представлява смартфон приложение, предназначено за решаване на математически задачи, включващи стереометрия в тях. Целта е да се използват възможностите на мобилното устройство за симулация на виртуална заобикаляща среда, чрез която чертаенето и решаването на подобен тип задачи да могат да се пренесат в триизмерното пространство.

Проблемът, който приложението решава е невъзможността да се визуализира триизмерно тяло върху двуизмерна равнина достатъчно ясно, например чертеж върху лист хартия. Независимо чрез каква проекция представяме триизмерния образ винаги компенсираме определен аспект, което понякога прави пространственото възприемане на чертежа доста трудоемка задача.

**Пример:** В следния чертеж на вписана пирамида в куб, независимо от проекцията са нужни поне няколко секунди фокусиране и пространствено визуализиране, докато успеем да си представим изкривената форма на пирамидата.

**C1**

*Фиг 0.1 Чертеж в ортографска проекция*

**C**

**А**

**B**

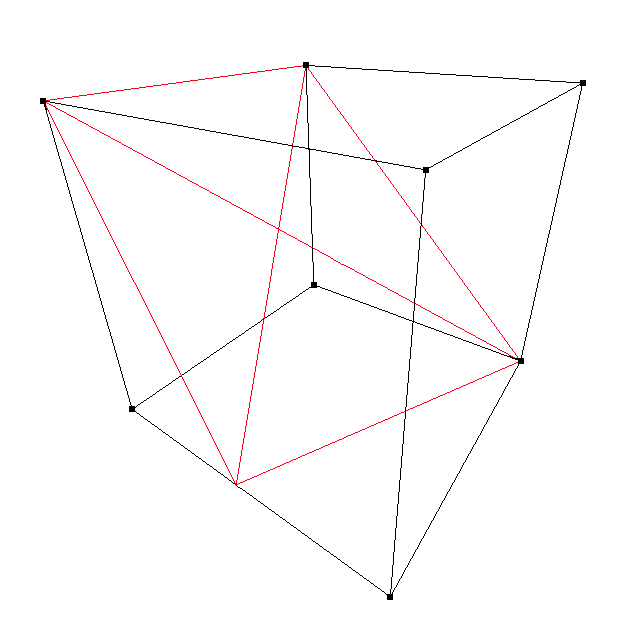
**D1**

**C1**

**D**

**А1**

**B1**



*Фиг 0.2 Чертеж в перспективна проекция*

**B**

**D1**

**B1**

**А1**

**D**

**C**

**А**

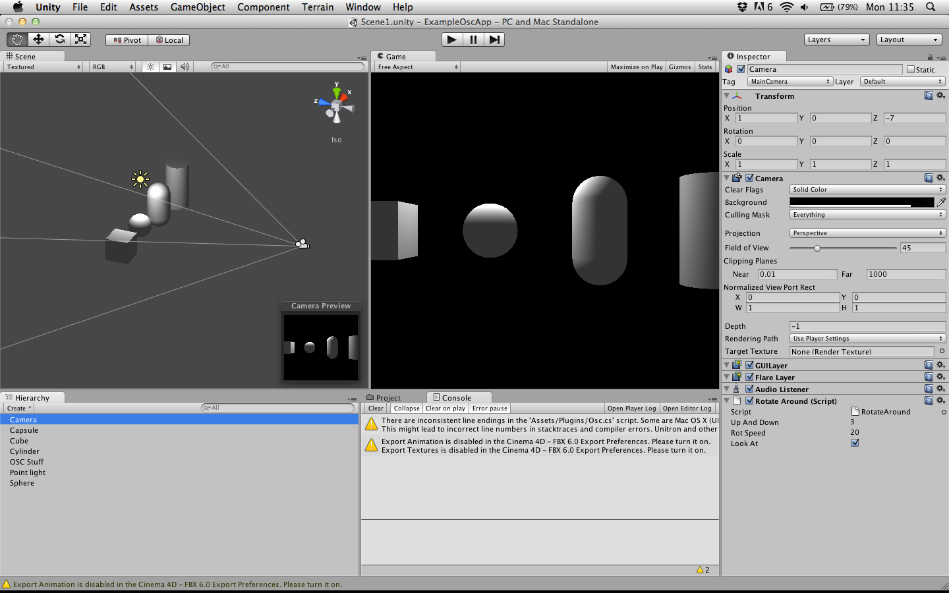
“Axyno” е единственото приложение към момента на пазара, което представлява опростен редактор на триизмерни обекти, създаден специално за използване със смартфон във виртуална среда. Приложението може да бъде използвано в училища и университети за елиминиране на проблемите при начертаването на една задача и акцентиране върху самата логическа последователност за решаването ѝ.

Работата с приложението става посредством отделен headset, предназначен за смартфони, като за целите на разработката беше използван Google Cardboard.

**ПЪРВА ГЛАВА**

Методи и технологии за реализиране на мобилни приложения

1. **Основни принципи, технологии и развойни среди, използвани при реализацията**
   1. **Технологии**
      1. **Unity Engine**





*Фиг 1.2 Потребителския интерфейс на Unity*

*Фиг 1.1 Unity лого*

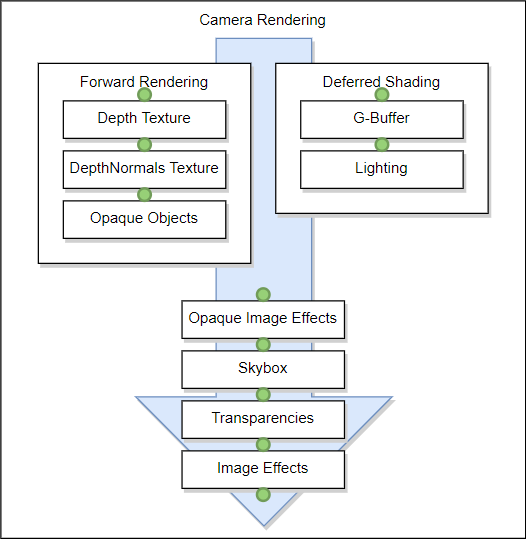
Unity („Юнити“) представлява широко-разпространен междуплатформен игрови двигател („гейм енджин“/„game engine“), използван за разработката на видеоигри, симулации, анимации и архитектурни дизайни. Чрез него могат да се създават 2D и 3D приложения, а крайния продукт може да бъде експортиран за някоя от 27-те различни поддържани платформи. Програмният език е C#, а за целите на разработка на смартфон приложение - Unity поддържа следните графични библиотеки: Vulkan за Android, Metal за iOS и OpenGL ES за двете.

Реализацията чрез Unity има своите предимства и недостатъци, като сред главните предимства са:

* Опростен и удобен потребителски интерфейс за работа с триизмерни обекти както и възможност за тестване във всеки един момент;
* Приложението се оптимизира спрямо платформата от самия енджин;
* Графиките са приспособими за всеки екран и устройство без никаква деформация или компромис с качеството;
* Изключително лесна интеграция/вграждане на поддръжка за виртуална реалност.

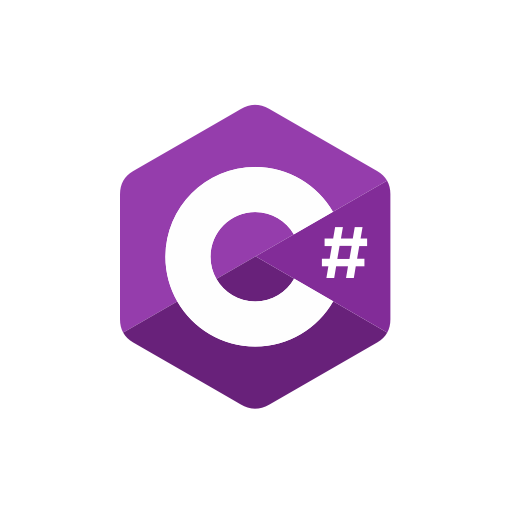
В Unity поведението на определен игрален обект се управлява от закачени за него „компоненти“. Въпреки че има широк набор от вече готови такива може да е необходимо по-персонализирано управление. В такива случаи може да бъде създаден допълнителен собствен компонент с помощта на външен скрипт. Скриптът бива изпълнен от съответния интерпретатор, който го превежда от символен на машинен език, пропускайки стъпката по генериране на бинарен файл, както правят например компилаторите.

Unity върши трудната работа по процеси като рендериране и оптимизиране, държейки ги далече от програмиста. По този начин вниманието и усилията му са насочени към бизнес логиката и естетическия вид на приложението.



*Фиг 1.3 Процеса на рендериране на сцена в Unity*

* + 1. **C#**



*Фиг 1.4 C# лого*

C# или C Sharp („Си Шарп“) е обектно-ориентиран език за програмиране, разработен от Microsoft, като част от софтуерната платформа „.NET“. Unity предлага първичен приложно-програмен интерфейс на C#, както за редактора под формата на различни плъгини, така и за самите разработвани приложения. Езикът е проектиран да балансира мощност с възможност за бърза разработка.

Едно от най-големите предимства на софтуерната платформа “.NET”, която Unity използва, е интегрираното автоматично управление на паметта. За това се грижи т.нар. система „garbage collector“, която е част от CLR (Common Language Runtime) компонента на платформата. Основната задача на тази система е да следи кога заделената памет за променливи и обекти вече не се използва и да я освобождава. Това освобождава програмиста от следене на използваните ресурси и проблеми като memory leaks („теч на памет“).

* 1. **Мобилни платформи**
     1. **Android**



*Фиг 1.5 Android лого*

Android („Андроид“) е операционната система на Google Inc. за мобилни устройства. Базирана е на Linux ядрото, като последната стабилна версия до момента е 9.0, именувана „Pie“. Unix-подобната операционна система е с open source (или „отворен код“) модел като повечето устройства съдържат и софтуер със затворен код.

За развитието на операционната система се грижат относително голям брой програмисти и софтуерни разработчици, които създават така наречените „Apps“ (от Applications - приложения), които представляват малки програми, целящи да разширят функциалността на системата. Тези приложения могат да бъдат сваляни от Интернет или от официалния магазин на Android, наречен Google Play, като приложенията от там могат да бъдат както безплатни, така и платени. Процеса по публикуване на приложение в магазина е доста лесен и общодостъпен – единственото условие е разработчика да има направен Google профил. Самия акаунт също е безплатен и за разлика от други мобилни магазини няма нито закупуване на лицензи, нито сертификационни тестове и периоди.

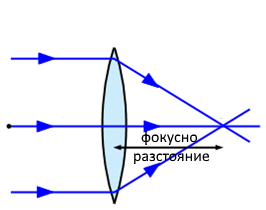
* + 1. **(iOS)**

(optional for now)

* 1. **Хардуер**
     1. **Google Cardboard**

*Фиг 1.6 Google Cardboard с поставен смартфон*

Google Cardboard е мобилна платформа за виртуална реалност, разработена от Google Inc. За разлика от други очила за виртуална реалност Google Cardboard използва процесорната мощност на смартфон за да симулира виртуална среда. Следенето на движенията и завъртанията също се осъществява от телефона, благодарение на вградените в него сензори - акселерометър и жироскоп.

 За да може образа на екрана да бъде чист и ясен продуктът използва двойно изпъкнали (биконвексни) лещи, поставени пред всяко око с фокусно разстояние от 45мм.

*Фиг 1.7 Действие на двойно изпъкнала леща*

Google предоставя три комплекта за разработка на софтуер (SDK – System Development Kit) за платформата си: един за операционната система Android, използващ Java, един за игровия двигател Unity, написан на C#, и един за операционната система iOS.

Понеже приложенията, проектирани за Google Cardboard биват пускани на мобилно устройство, след което самото то бива поставено в определеното за целта място, нямаме пряк достъп до сензорния екран на телефона. Затова интеракцията с приложението става посредством бутон, разположен в горния десен ъгъл на устройството, който притиска кондуктивно парче до екрана регистрирайки допир.



*Фиг 1.8 Местоположение на бутона за интеракция*

**ВТОРА ГЛАВА**

Проектиране на структурата на Android мобилни приложения за виртуална реалност

1. **Функционални изисквания към Android приложения**
   1. **Изисвания към изграждането на логиката**

Трябва да бъде изградена осносна структура на приложението. Да се направят основни класове чрез които добавянето на нови функционалности да става бързо и лесно. Да се изгради лесна и логически-издържана комуникация между различните части на приложението.

* 1. **Изисквания към графиката на приложението**

Приложението трябва успешно да симулира виртуална заобикаляща среда. Всеки текст, картинка и символ трябва да бъде ясно разграничаван и лесно разбираем. Графиката на приложението не трябва да забавя изпълнението на програмата, като кадрите в секунда е желателно да не падат под 90. Чертежът трябва да е с неограничена големина без това да се отразява директно на производителността.

* 1. **Потребителски интерфейс и итерация с потребителя**

Потребителският интерфейс на приложението трябва да бъде максималко опростен и разбираем. Цялата навигация ще трябва да се осъществява само чрез Google Cardboard визьора, затова всички менюта и бутони трябва да са с достатъчно голям размер че да не бъде изпитвано затруднение при използването им.

* 1. **Функционалност**

Приложението трябва да изисква възможно най-малко права от устройството. Данните, прочетени от акселерометъра и жироскопа на смартфона, трябва да бъдат извличани и обработвани достатъчно бързо, подсигурявайки следене на въртенето на главата в реално време.

* 1. **Производителност**

Приложението не трябва да забива или замръзва без причина. Моменти на насичане, прекъсване или други дефекти не трябва да бъдат забелязвани. Също така латентността трябва да бъде минимална през цялото време.

1. **Съображения за избор на програмни средства и развойната среда**
   1. **Игрови двигател Unity**

За реализацията на приложението беше взет избора да се използва игрови двигател поради многото си предимства при разработка, като например:

* Повечето от сложните и времеемки аспекти от приложението като управление на паметта, зареждане на асети, изчисляване на осветлението и др. са вече написани и готови за ползване;
* Повечето игрални двигатели са междуплатформени, което прави експортирането на приложение за специфична платформа доста лесна задача

Игровият двигател, който беше избран е Unity. Причината е че инструментите му са познати, тъй като вече различни проекти са били правени ползвайки него както за училище, така и за хоби. Това пропуска стъпката от заучаване на нови технологии и увеличава продуктивността.

(СРАВЕНИЕ С ДРУГИ ТУК)

* 1. **Google Cardboard платформа**

Понеже приложението е проектирано като продукт, имплементиращ виртуална реалност, с намерение да се използва предимно в училища използването на смартфон беше очевидния избор за устройство. Google Cardboard платформата допринася за лесната портативност и общодостъпността на приложението. Тя е евтина и лесна за работа, като не се изисква по-специализиран хадруер за нея.

* 1. **Операционна система Android**

След обстойно проучване на пазара за мобилни операционни системи се стигна до заключението че Android е най-уместният избор поради следните причини:

* Наличие на физически устройства, на които да бъде тествано приложението;
* Не се изискват допълнителни финансови средства;
* Липса на опит с други платформи;
* Лесно и бързо публикуване в магазина за приложения Google Play;
  1.  **GitHub система за контрол и управление на версиите**

*Фиг 2.1 GitHub лого*

GitHub представлява уеб-базирана услуга за качване на софтуерни проекти върху отдалечен сървър (т.нар. хранилище) и осъществяването на съвместни разработки върху тях. Базиран е на Git системите за контрол и е избран заради вече съществуващия опит с него.

* 1.  **Trello мениджър на задачи**

*Фиг 2.2 Trello лого*

Trello е уеб-базирано приложение за управление на проекти и отделните задачи по тях. Използването му се състои от класифициране на подзадачите спрямо различни критерии и следене на всичко планирано до момента, както и визуализиране на прогреса при разработката.

* 1. **Notepad++ текстов редактор**



*Фиг 2.3 Notepad++ лого*

Цялостната процедура по писане на кода беше осъществена в Notepad++. То представлява опростен текстов редактор, разпространяващ се като софтуер с отворен код. Главното му предимство е малкият му размер – почти 23MB след инсталация, както и опростения му интерфейс и оцветяване на синтаксиса за над 50 програмни езика.

* 1.  **Paint.net графичен редактор**

*Фиг 2.4 Paint.net лого*

Paint.net е безплатен редактор на растерна графика, разработен на .NET платформата. От функционална гледна точка той предоставя много повече инструменти от редактора на изображения по подразбиране в Windows – “Paint“. Това, заедно с малкия му размер и високата производителност, го прави изключително добър за работа с малки и средни проекти.

**(blender if used at all when design comes to place)**

**+ идеи за бъдещо развитие**

**ТРЕТА ГЛАВА**

Програмна реализация на “Axyno”

1. **Архитектура на приложението**
   1. **Типове обекти за построяване**

Чертежът бива построен чрез три основни обекта, свързващи се помежду си. Той може да съдържа неограничен брой съставляващи го части.

* + 1. **Точка**

Точката представлява единична позиция в триизмерното пространство. Тя съдържа информация за съответните си координати по трите оси и в приложението бива репрезентирана като сфера.

Използва се за да обуславя определена линия. Към една точка могат да бъдат свързани безброй много линии.

* + 1. **Линия**

Линията е права, ограничена от две точки. Тя съдържа информация за двете си точки и бива визуализирана като дълъг цилиндър, свързващ ги.

Линията репрезентира определен ръб или права в чертежа, като между две точки може да съществува само една линия.

* + 1. **Ъгъл**

Ъгълът е геометричен обект, съставен от две прави с обща начална точка, наричана връх на ъгъла. Съхранява инфо за двете линии, съставляващи го и в приложението се представя като дъга от едната линия до другата.

Обектът се използва за да се визуализира числената големина на ъгъла, отразяващ степента на завъртане на едната линия около върха, така че тя да съвпадне с другата.

* 1. **Разделение на класовете**

Всеки инструмент, обект или действие в програмата си има отделен клас, определящ функциалността им.

|  |  |
| --- | --- |
| Инструмент/Действие/Обект | Съответстващ клас |
| Обект от тип ъгъл | AngleObject.cs |
| Управление на меню чрез бутон | Button.cs |
| Смяна на цвета на обект | ChangeColor.cs |
| Смяна на типа линия | ChangeLineType.cs |
| Построяване на ъгъл | CreateAngle.cs |
| Построяване на ъглополовяща | CreateBisector.cs |
| Построяване на окръжност | CreateCircle.cs |
| Построяване на линия | CreateLine.cs |
| Построяване на перпендикуляр | CreatePerpendicular.cs |
| Построляване на правилен n-ъгълник | CreatePolygon.cs |
| Изтриване на обект | Delete.cs |
| Избутване на форма в пространството | Extrude.cs |
| Именуване на обекти и задаване на стойностите им | Keyboard.cs |
| Обект от тип линия | LineObject.cs |
| Разделяне/Удължаване на линия | LineSplit.cs |
| Движение на обекти | MoveObjects.cs |
| Обект от тип точка | PointObject.cs |
| Завъртане на чертежа около ос | RotateButton.cs |
| Селектиране/Деселектиране на всички обекти | SelectOrDeselectAll.cs |
| Разделяне на линия на равни отсечки | SubdivideLine.cs |
| Избор на инструмент  *Фиг 3.1 Класовете, съответстващи на инструментите* | ToolButton.cs |

* 1. **Главни класове на приложението**
     1. **Tool.cs**

Основният клас, който всички действия наследяват е “Tool“. Това е абстрактен клас с няколко метода, които биват използвани от всички инструменти.

* + - 1. **Initiate**

Това е главният абстрактен метод, който бива презаписан от всеки наследяващ клас. Той бива извикван при натискането на съответния бутон за използване на инструмента.

* + - 1. **GetObjects**

Този метод връща списък, съдържащ обекти от чертежа. Чрез подаваните му агрументи може да се филтрира списъка по тип (точка, права, ъгъл) и условие дали дадените обекти да са селектирани.

* + - 1. **ReportMessage**

За извеждане на съобщение до потребителя се използва този метод. Той показва даденото в аргументите съобщение (най-често за грешка) като текст в сцената. След показването на съобщението се извиква скрития метод CleanMessage като корутина за изчистване на текста.

* + - 1. **CleanMessage**

Метод от тип IEnumerator и извикван чрез StartCoroutine(). Това позволява паузирането на изпълнението му за определен брой секунди. В случая метода изчаква пет секунди преди да изчисти текста в сцената.

* + - 1. **GetTaskTransform**

Връща трансформационния компонент на игралния обект „Task”. Това позволява на всеки нов обект да бъде прикачен като дъщерен обект към него. По този начин чрез манипулация на обекта „Task“ се осъществява манипулация на всички обекти, съставляващи чертежа.

* + - 1. **Vibrate**

При извикването на Vibrate() се включва вибрацията на мобилното устройство за къс период от време. Това позволява на потребителя да получава тактилна обратна връзка от натиснати бутони, използвани действия и др.

* + 1. **CreatedObject.cs**

Основен абстрактен клас за всички обекти, съставляващи чертежа (точки, линии, ъгли). Съдържа методи и променливи присъщи за всеки обект.

* + - 1. **ChangeColor**

Метод, който променя цвета на материала на обекта с цвят, подаден в аргументите.

* + - 1. **Select**

Скрит метод, който селектира обекта, правейки го участник във всички последващи манипулации. Променя стойността на isSelected променливата на true и сменя шейдъра на обекта на Silhouette-Outlined Diffuse за визуална репрезентация на селектиран обект.

* + - 1. **Deselect**

Също скрит метод, който изключва обекта от това да бъде избран. Поставя променливата isSelected във false и връща обратно Diffuse/Specular шейдъра на обекта.

* + - 1. **SelectClick**

Публичен метод, който следи ако при извикването си сме със инструмента за избиране („Select Tool“) и извиква Select() или Deselect() в зависимост от стойността на isSelected на обекта.

* 1. **Допълнителни класове**
     1. **PointObject**

Всяка точка от чертежа съдържа в себе си класът PointObject. Той съдържа помощни методи, благодарение на които се управлява поведението на обекта. В него се съдържат следните методи:

* + - 1. **Start**

При създаването на точка се проверява да не би тя да е на позицията на друга. За да се избегне интерсекция на два обекта, позицията на всяка точка се дефинира чрез центъра ѝ плюс радиуса на сферата, репрезентираща точката. Ако има друга точка в тази позиция (Фиг. 3.2), двете точки се сливат в една, като ако има прави, тя се прекарват през тази точка (Фиг. 3.3).



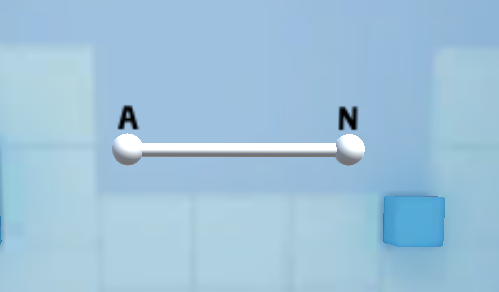
*Фиг 3.3*

*Фиг 3.2*

* + - 1. **ConnectedTo, Connect, Disconnect**

Това са различни методи, осигуряващи свързването на точка и права. Те съответно свързват, разкачат и проверяват нали е закачена дадена права за точката. Една точка може да бъде свързана към неограничен брой прави.

* + - 1. **AddText, GetText**

Точките могат на бъдат именувани с имена, състоящи се от максимум 4 символа от латинската азбука. По този начин могат да бъдат описвани ъгли и форми (Фиг. 3.5). Тези методи съответно поставят и връщат името на точката.

*Фиг 3.4 Правата AN е определена от т.A и т.N*

* + 1. **LineObject**

За обекти от типа линия се изплозва класът LineObject. Всяка линия бива точно определена от две различни точки. Поведението на тези обекти се осъществява чрез следните методи:

* + - 1. **SetPoints, OnDestroy**

Точките, към които е свързана линията биват запазени в класа като при изтриването на линията се изтриват и те, ако това е единствената права към която са свързани.

* + - 1. **ChangeColor, SetTransparency**

На всяка линия освен цвета, може на и бъде сменен типа. За това се грижат тези два метода. (За повече инфо т.3.3.2 и т.3.3.3)

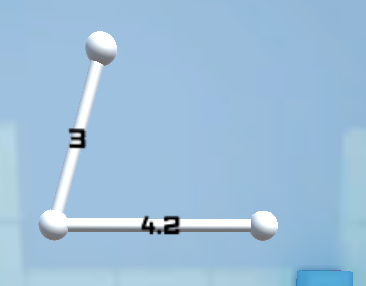
* + - 1. **ConnectAngle, DisconnectAngle**

Към всяка линия могат да се свържат неограничен брой ъгли. Тези методи съответно закачат и разкачат определен ъгъл на линията.

* + - 1. **UpdatePosition**

Понеже формата на линията зависи от двете си точки при всяка тяхна манипулация, дължината и позицията на линията се променя. Метода UpdatePosition се грижи за правилното преизчисляване на линията.

* + - 1. **AddText**

****На линиите може да им бъде давана определена репрезентативна стойност. Тя може да съдържа както целочислени, така и числа с десетична запетая.

*Фиг 3.5 Линии с различна дължина*

* + 1. **AngleObject**

В класа, представляващ ъгъла се поместват методи, управляващи действието на обект от типа ъгъл. Някои от тези методи са:

* + - 1. **Connect, SwitchLine**

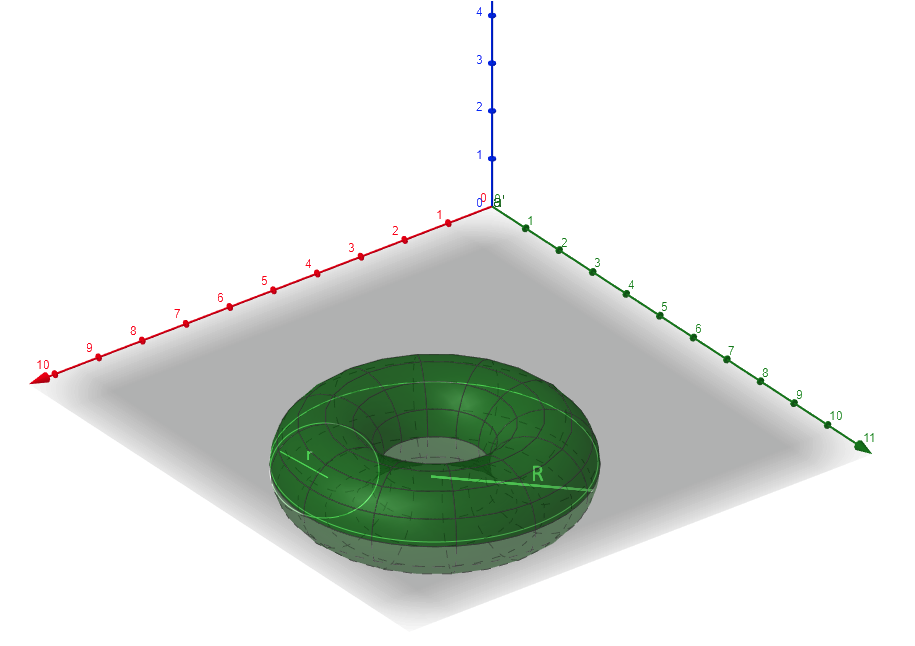
Методи, използвани за управление на линиите, обуславящи ъгъла. Съответно свързват определени линии към ъгъла или ги сменят.

* + - 1. **UpdateAngle**
      2. **BuildTorus**

Метод, на който му се подават брой сегменти от 32-сегментов тор и съответната форма, репрезентираща ъгъла бива построена по следния начин:

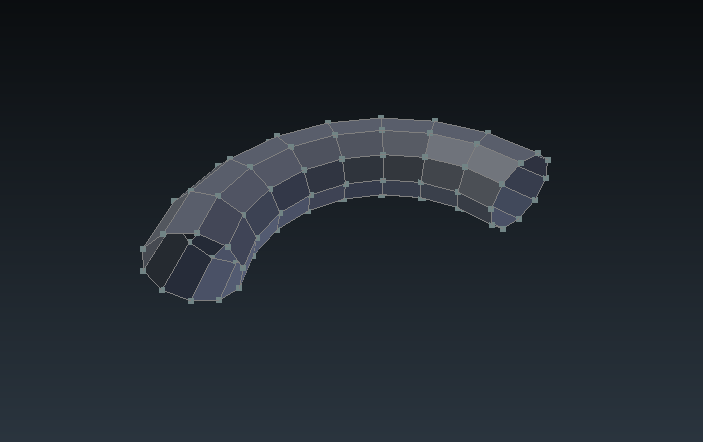
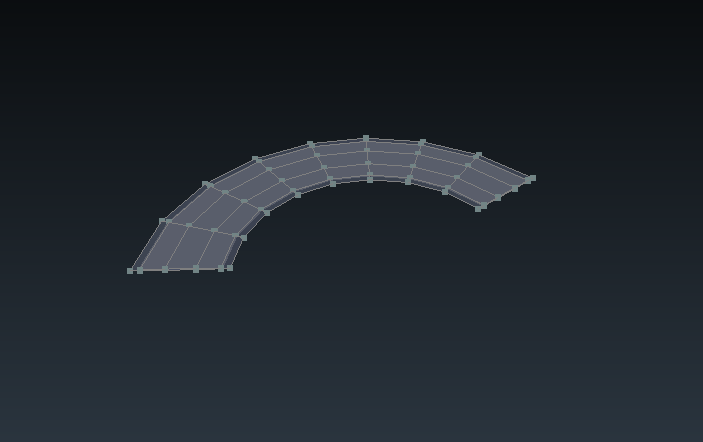
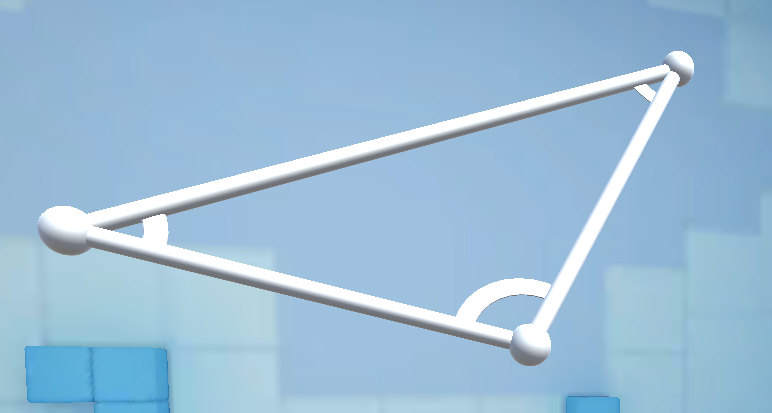
Първоначално се избират параметрите на формата (Фиг. 3.6), които включват:

* Радиус на окръжността (r)
* Радиус на завъртане (R)
* Брой на точките, обуславящи една окръжност (в случая малката окръжност е съставена от 10 точки, но в приложението това число е 12)
* Брой на окръжностите, обуславящи торът (в случая той е съставен от 12 окръжности, но в приложението това число е равно на 32)



*Фиг 3.6*

След това се построява единичната окръжност с радиус r и се завърта толкова пъти по окръжността, съставляваща тора, колкото е броя на подадените сегменти. По този начин се създава част от тор. (Фиг. 3.7) Тази форма бива „сплескана“ така че да заприлича повече на двуизмерна дъга (Фиг. 3.8). След финално позициониране и завъртане ъгловата форма се поставя на мястото си (Фиг. 3.9).

Броят на сегментите от които ще бъде съставен торът, репрезентиращ ъгъла, се изчислява от фактът че моделът е със зададен максимален брой сегменти 32, като това се равнява на 360°. Изписано като формула, това изглежда така:

*Фиг 3.9 Ъгли с различна големина в триъгълник*

*Фиг 3.8*

*Фиг 3.7*

* 1. **Карта на всички класове**

**(to be done when coding is finished)**

* 1. **Интеграция с Cardboard SDK**

От Cardboard комплекта за разработка на софтуер („Software Development Kit“), предназначен за Unity, бяха включени три обекта в приложението:

* + 1. **GvrEventSystem**

Този обект съдържа в себе си класа GvrPointerInputModule, който се използва заедно с GvrBasePointer за да осъществи имплементация на основния клас в Unity за потребителска интеракция – BaseInputModule. GvrInputModule осигурява тази реализация, така че елементите на потребителския интерфейс на базата на Canvas(uGUI) и обектите в 3D сцената да могат да бъдат взаимодействани с VR приложение.

* + 1. **GvrEditorEmulator**

По време на процеса по разработката на приложението всяко тестово пускане трябваше да бъде от мобилно устройство. Но процеса по качване на приложението на смартфон всеки път беше доста времеемка задача. Затова обектът GvrEditorEmulator позволи емулация на Android устройство директно в Unity. Чрез основни контроли като Alt + движение на мишката се симулира въртене на устройството, а чрез натискане на левия бутон – допир до сензорния екран.

* + 1. **GvrReticlePointer**

Този клас наследява от GvrBasePointer, като добавя графична репрезентация на функциалността му. Начертава кръг в центъра на екрана, симулиращ курсор. Кръга се разширява ако може да се осъществи взаимодействие с гледания обект.

*Фиг 3.2 Потребителския курсор*



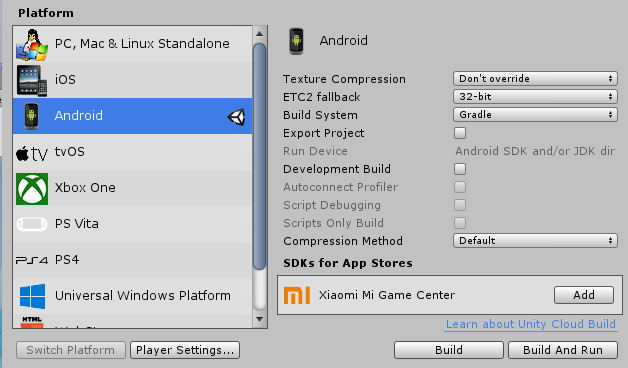
*Фиг 3.3 Курсора, насочен към обект*

* + 1. **Допълнителни настройки**

За да работи правилно приложението, трябва да бъде направена и допълнителна конфигурация, включваща:

* + - 1. **Избор на платформа**

За да може приложението да бъде коректно експортирано за Android платформата е нужно това да бъде посочено в настройките.

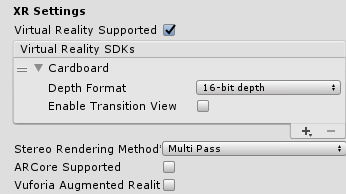


*Фиг 3.4 Менюто за избор на платформа*

Чрез смяната на платформата чрез File > Build Settings игровия двигател предприема всички нужни действия и оптимизации за успешното експортиране на приложението към Android платформата.

* + - 1. **Поддръжка за виртуална реалност**

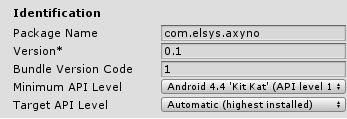
След импортиране на пакета, предназначен за разработване на приложения за Google Cardboard, от менюто за настройки на разширената реалност трябва да бъде селектирано поддържането на виртуална реалност (Virtual Reality Supported), а след това да бъде избрано „Cardboard“ като пакет. Менюто се намира на Edit > Project Settings > Player Settings > SR Settings



*Фиг 3.5 Панела за настройки на разширената реалност*

* + - 1. **Поддръжка за максимално ниска версия на Android**

Cardboard пакета изисква минимално Android API ниво 19. Това се равнява на Android версия 4.4. Следователно приложението ще бъде поддържано от всички Android устройства с версия на операционната си система равна или по-висока от нея. Тази настройса се променя от Edit > Project Settings > Player Settings > Other Settings

****

*Фиг 3.6 Панела за различни идентификационни настройки*

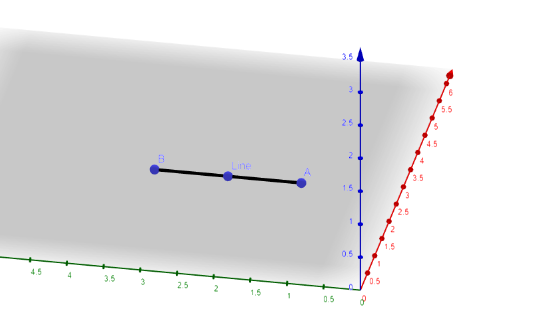
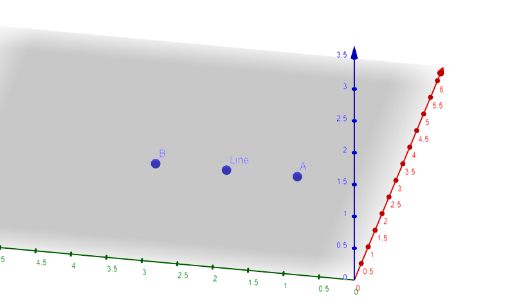
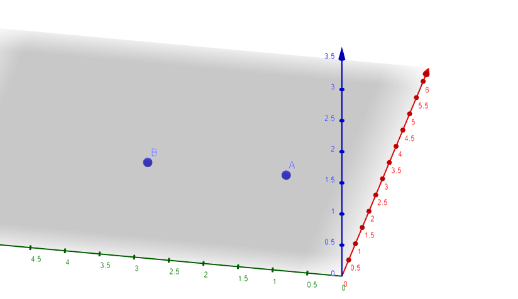
1. **Графика на приложението**
   1. **Потребителски интерфейс**
   2. **Графично представяне на елементите (shaders n such)**
2. **Реализация на инструментите и действията**
   1. **On-screen buttons**
      1. **Отваряне на меню**
      2. **Избор на инструмент**
      3. **Завъртане на чертежа**
      4. **Клавиатура**
      5. **Създаване на обекти**
   2. **Инструменти**
      1. **Select Tool**
      2. **Rotate Tool**
      3. **Shape Select Tool**
   3. **Действия**
      1. **Build Line / Постояване на линия**

В класа за построяване на линия се помещават два метода:

* + - 1. **BuildLine**

Получава две точки като аргумент и построява линия между тях по следния начин:

Първо се създава линия с дължина нула на позицията на едната точка и се измерва разстоянието между точките (Фиг. 3.7). След това линията бива транслирана по вектора, определен от двете точки, заставайки точно по средата им (Фиг. 3.8). Накрая дължината на линията бива зададена да е равна на разстоянието между точките. (Фиг. 3.9)

****

*Фиг 3.9*

*Фиг 3.7*

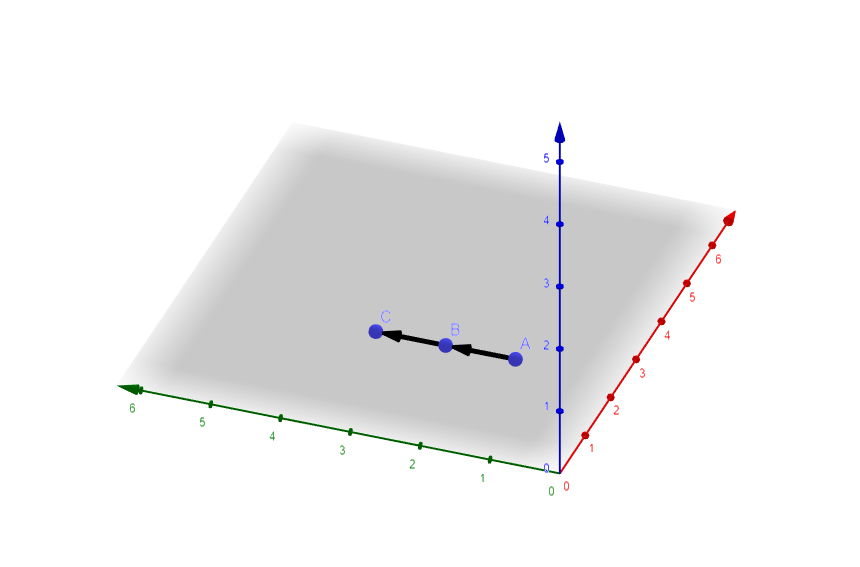
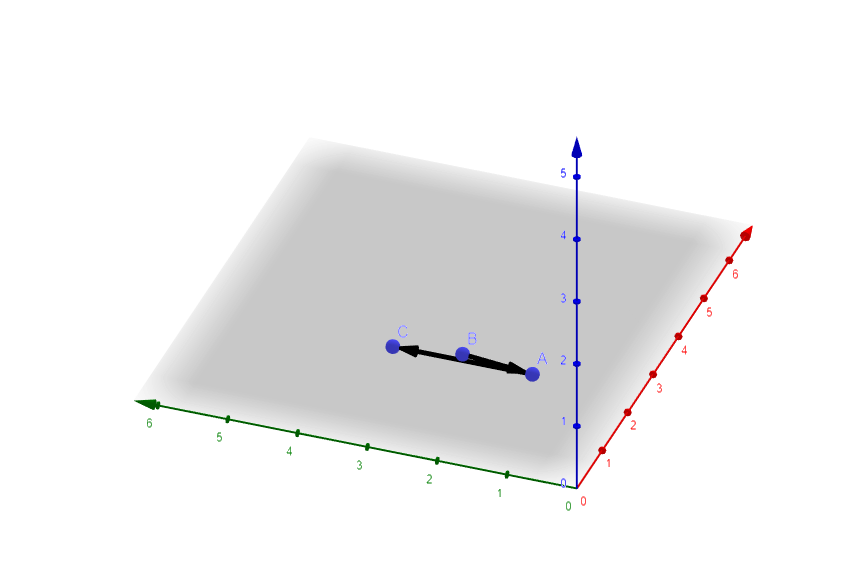
*Фиг 3.8*

* + - 1. **IsBetween**

Този метод проверява дали дадена точка се намира между други две, ако и трите лежат на една линия. Той не се използва при построяването на линия, но бива извикван от класовете, наследяващи CreateLine.

Използва се факта че скаларното произведение на два нормализирани вектора е равно на 1, ако векторите имат една и съща посока, или на -1 ако сочат в противоположни посоки.

Ако искаме да проверим дали точка C лежи между т.A и т.B, трябва да проверим дали скаларните произведения . и . връщат -1 (векторите биват нормализирани предварително).

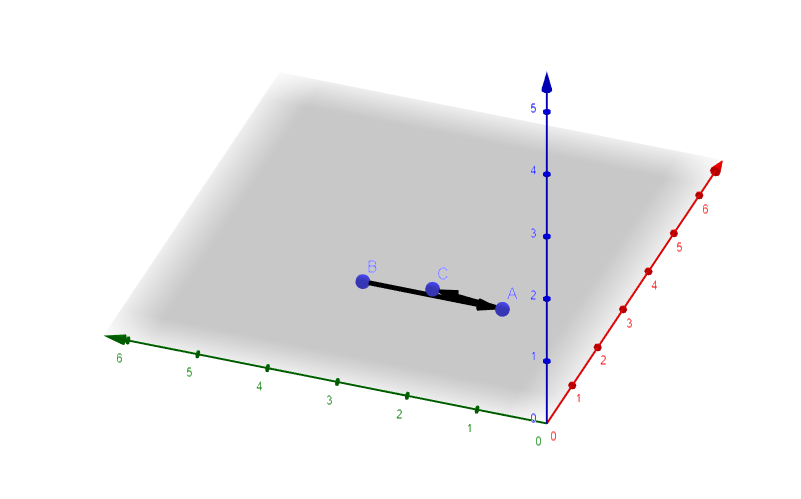
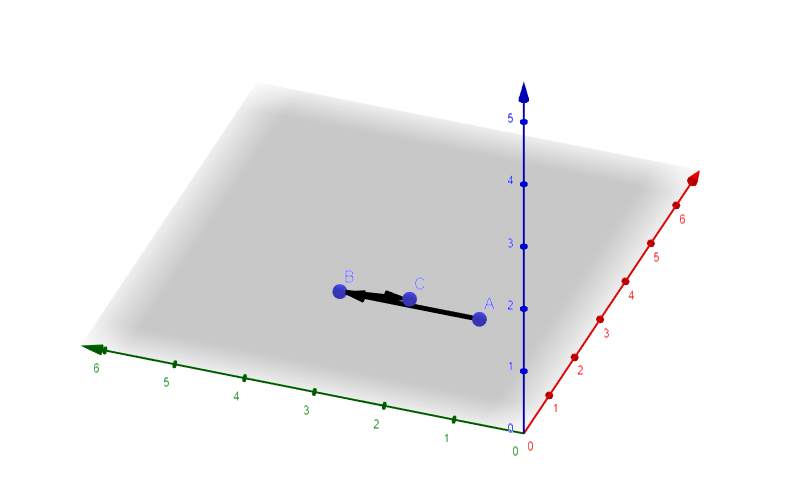
Например ако точката не лежи между другите две:

*Фиг 3.11*

*Фиг 3.10*

Произведението на и ще бъде 1, защото и двата вектора имат една и съща посока (Фиг. 3.10), докато това на и ще бъде -1, заради противоположните им посоки (Фиг. 3.11).

Докато ако т.C лежи между т.A и т.B:

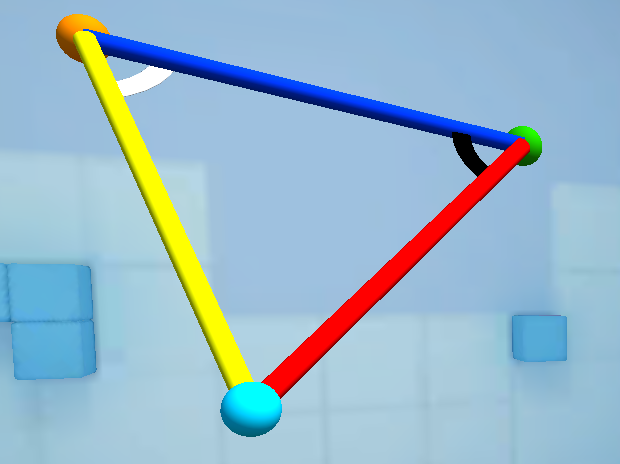


*Фиг 3.13*

*Фиг 3.12*

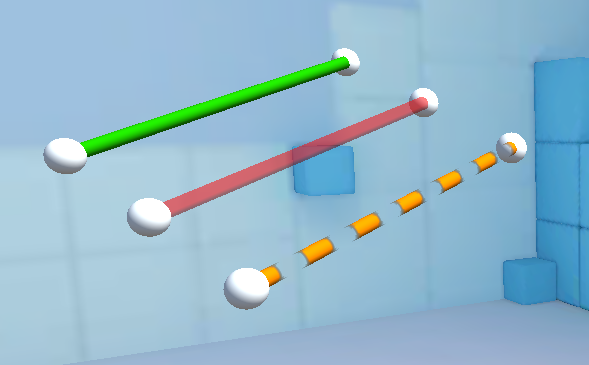
Векторите и ще бъдат с противоположни посоки (Фиг. 3.12), както и и (Фиг. 3.13). Следователно и двете скаларни произведения ще бъдат -1.

* + 1. **Change Color / Смяна на цвета**

На всеки един обект от чертежа може да му бъде сменен цвета. След избор на някои от осемте цвята материалите на всички селектирани обекти ще придобият съответния цвят.

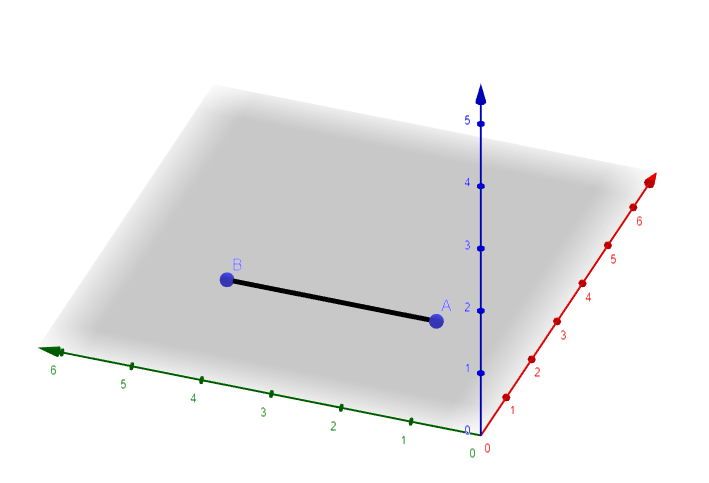
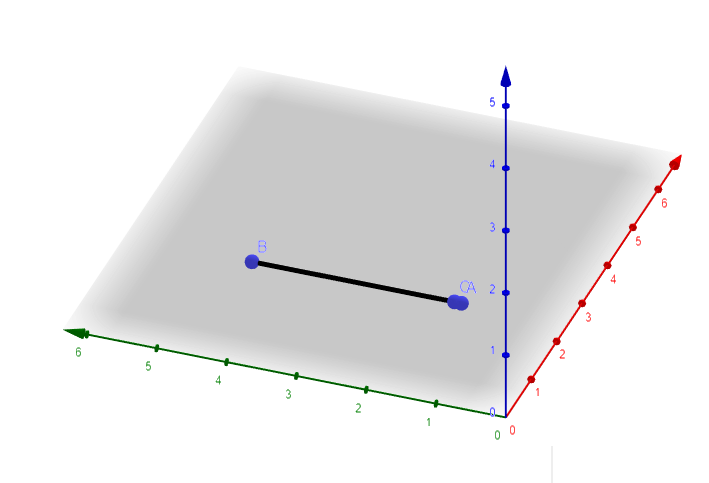
*Фиг 3.14 Триъгълник с различно оцветени компоненти*

* + 1. **Change Line type / Смяна на типа линия**

На всяка една линия може да бъде променен типа, без това да се отрази на цвета ѝ. Потребителя може да избира измежду три варианта: солидна, прозрачна и прекъсната

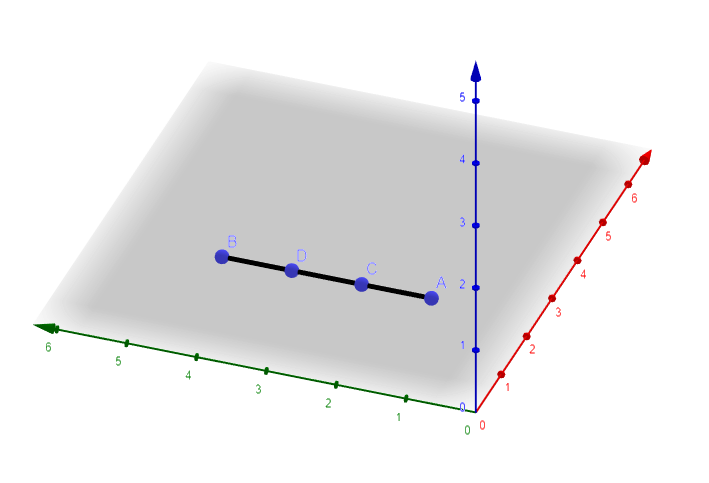
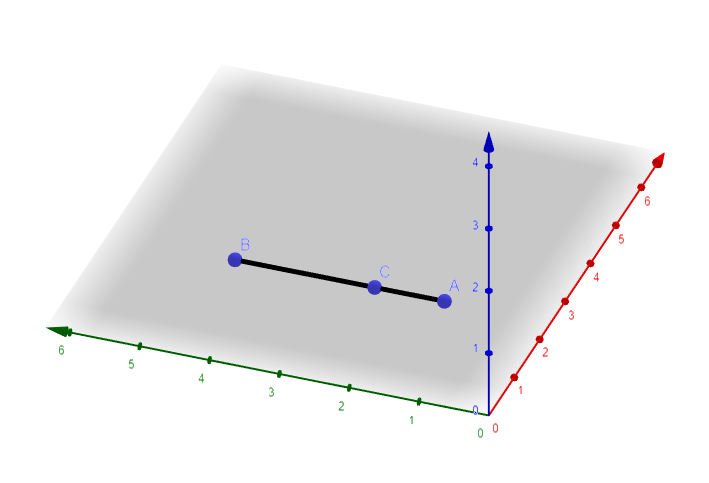
*Фиг 3.15 Линии от различен тип и цвят*

* + 1. **Subdivide / Деление на линия**

Инструментът се използва за разделяне на линия на няколко еднакви части. Диапазона е от 2 до 10 равни сегмента. След избиране на бройката линията ще бъде разделена чрез създаването на нови точки на съответните места и последователното им свързване. Например ако искаме да разделим линията AB (Фиг. 3.16) на три еднакви части програмата предприема следните действия:

*Фиг 3.17*

*Фиг 3.16*

Първоначално бива изчислена дължината на всеки сегмент чрез формулата . След това се създава нова точка C на позицията на т.А (Фиг. 3.17). Тя бива транслирана по линията на разстояние (Фиг. 3.18) като номерата на точките започват от 1 и се увеличават с единица за всяка следваща. Процеса се повтаря за пъти докато накрая линията не бъде разделена точно (Фиг. 3.19).

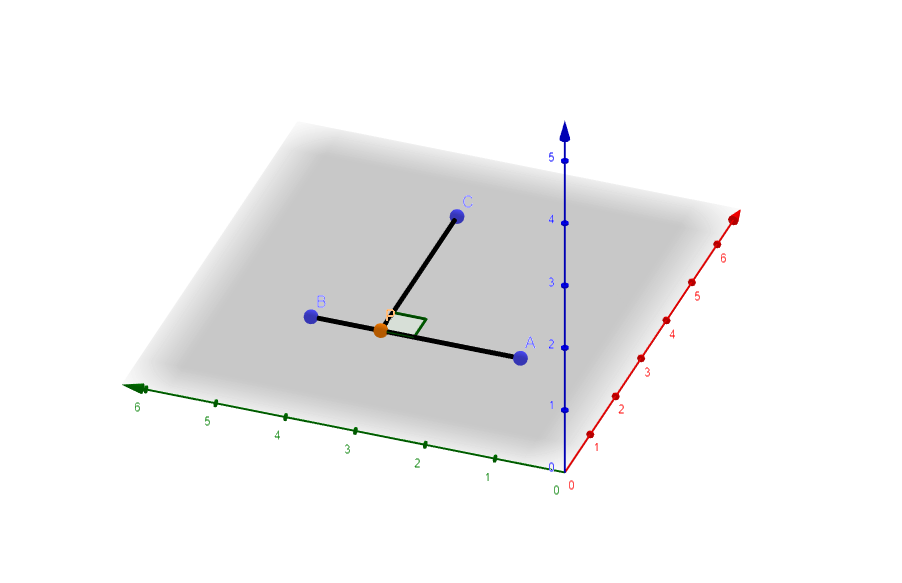
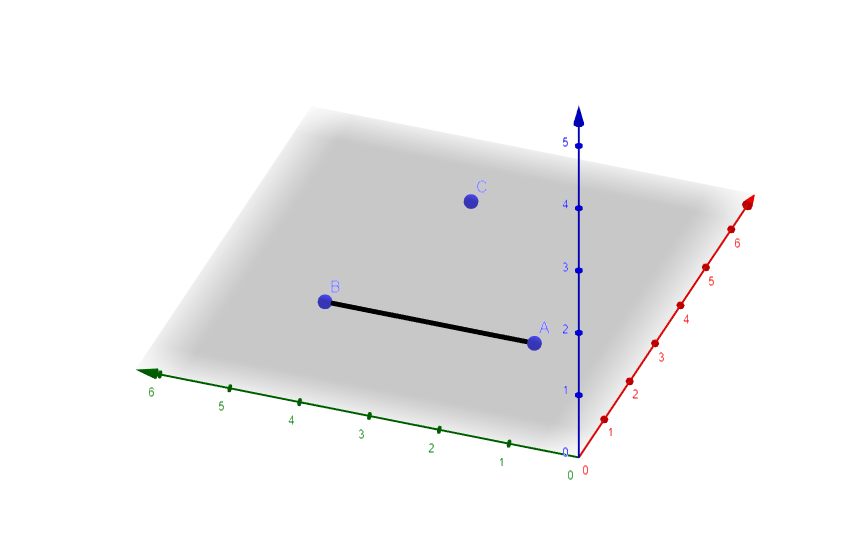
*Фиг 3.18*

*Фиг 3.19*

Последната стъпка е да се изтрие старата линия и да се създадат нови линии между новосъздадените точки.

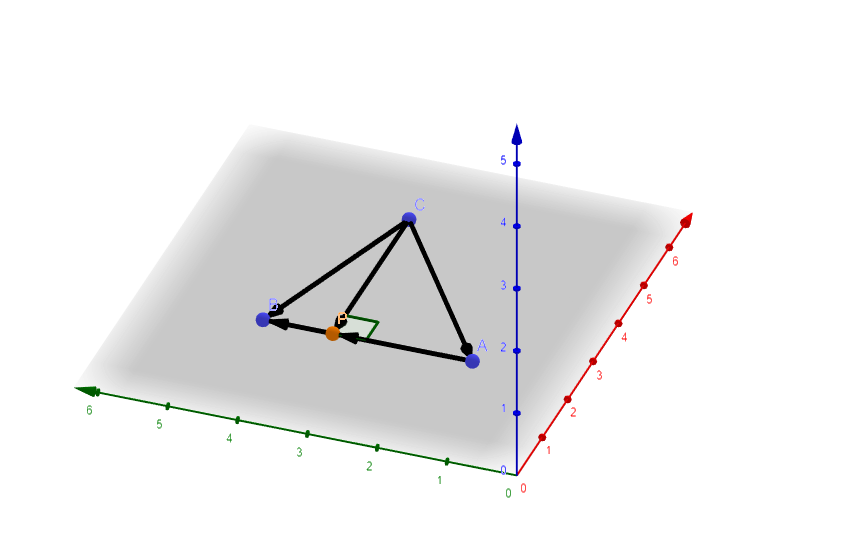
* + 1. **Build Perpendicular / Строене на перпендикуляр**

При селектирана точка и права от чертежа може да бъде пуснат перпендикуляр. Процесът по изграждането ѝ изглежда по следния начин:

**** Ако имаме права AB и точка C, нележаща на нея (Фиг. 3.19) можем да построим само една права, перпендикулярна на AB, минаваща през т.C. Ако знаем координатите на всички точки, можем да намерим и съответната позиция на точка P от правата, която свързана с т.C ще ни даде перпендикуляр (Фиг. 3.20)

*Фиг 3.20*

*Фиг 3.19*



*Фиг 3.21*

Нека първо представим всички прави като вектори (Фиг. 3.21) От там можем да изведем формулата

Но лежи върху , следователно

където като променяме местим т.P по правата AB. При заместване на в горната формула получаваме че

Оттук вече можем да изведем формула за координатите на точка P, която изглежда така:

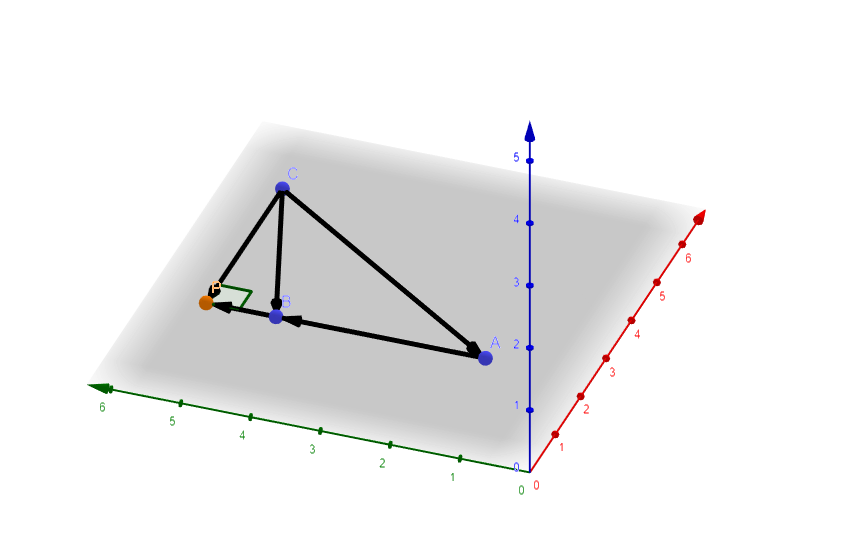
След съкращаване на съответните координати на точка C от двете страни, финалната формула става

Освен всичко това знаем, че и са перпендикулярни. Това означава че скаларното им произведение е равно на нула. Разписано това изглежда така:

Ако заместим координатите на точка P с тези в горната формула ще получим уравнение за .

След изнасяне само на получаваме финалния израз за константата, който е

Изчислявайки го, вече може да заместим в горната формула и да получим точните координати на точка P.

Ако точката се намира между т.А и т.B, то линията бива разделена на две, чрез т.P. Иначе ако петата на перпендикуляра се пада извън правата, тя бива удължена до точката (Фиг. 3.22).

*Фиг 3.22*

* + 1. **Create Angle / Построяване на ъгъл**

За създаването на обекти от типа ъгъл, се грижи класът CreateAngle. При селектиране на две линии от чертежа, се създава обект от типа ъгъл. След това новосъздадения ъгъл бива свързан към линиите и му бива зададена определената форма и позиция чрез UpdateAngle метода му.

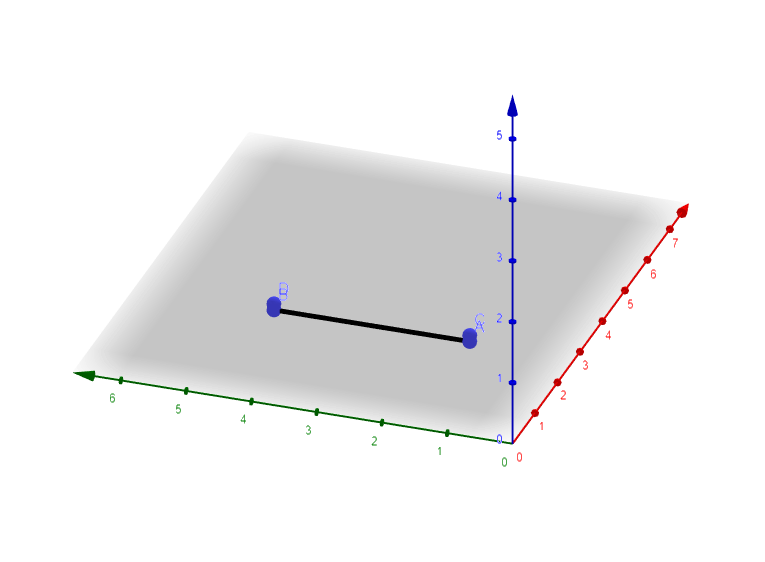
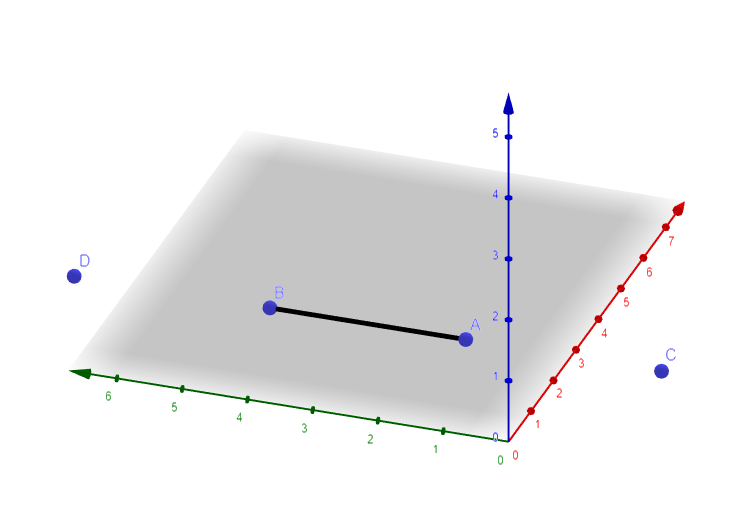
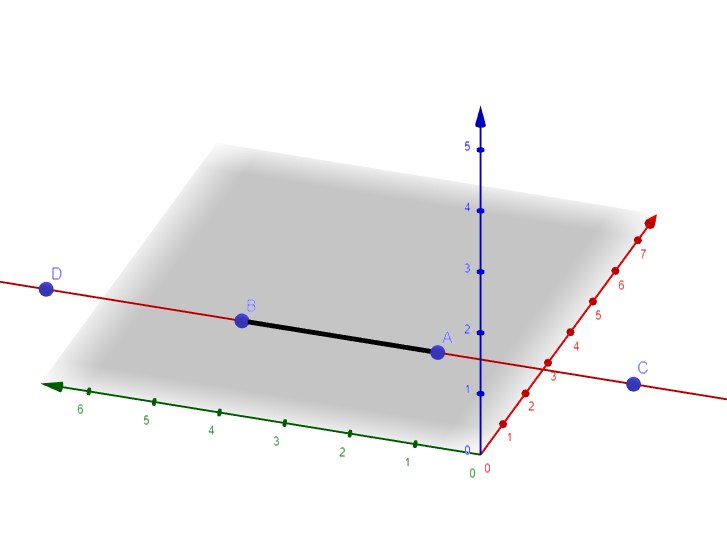


*Фиг 3.23 Построени ъгли с различни цветове*

* + 1. **Expand Line / Разширение на линия**

Приложението има инструмент, който съчетава две функционалности в едно: удължаване на права и портрояване на нова точка върху права. Класовете, който отговарят за това са LineSplit и LineHover. За осъществяването на тези действия се създава помощна линия по следния начин:

* + - 1. **LineSplit**

Ако искаме да построим произволна точка върху AB (или съответно да удължим AB) първо се създават две нови временни точки (т.C и т.D) на местата на същите позиции като точките, ограничаващи правата (Фиг. 3.24). След това тези точки биват транслирани извън линията по вектора, определен от точките на правата (съответно за т.C и за т.D) на определено разстояние. Това разстояние е достатъчно голямо, така че новосъздадените точки да не се забелязват в сцената. (Фиг. 3.25) Нова специална спомагателна линия бива построена между точки C и D, която вместо LineObject съдържа LineHover класа в себе си. (Фиг. 3.26)

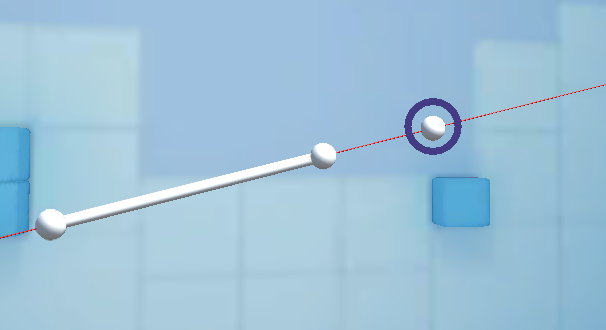
*Фиг 3.26*

*Фиг 3.25*

*Фиг 3.24*

* + - 1. **LineHover**

Този клас се ползва само и единствено от спомагателната линия. При местене на курсора по правата може предварително да се види позицията на бъдещата точка (Фиг. 3.27).



*Фиг 3.27*

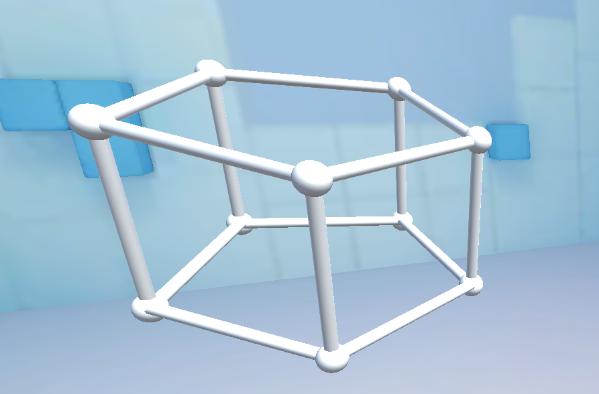
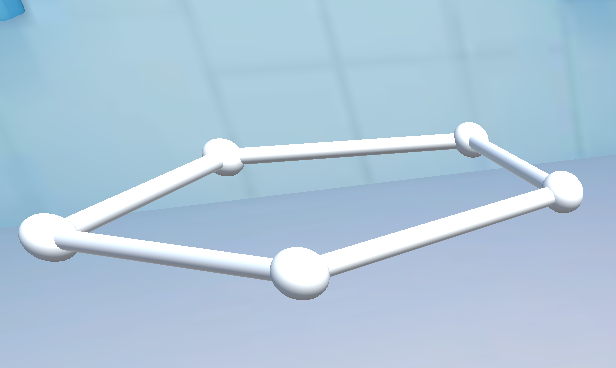
След потвърждение от страна на потребителя, новата точка бива създадена и ако позицията ѝ е извън правата бива свързана с по-близката от двете точки, ограничаващи линията. Ако точката се падне да лежи на линията, самата линия бива заменена от две прави, свързващи трите точки една за друга.

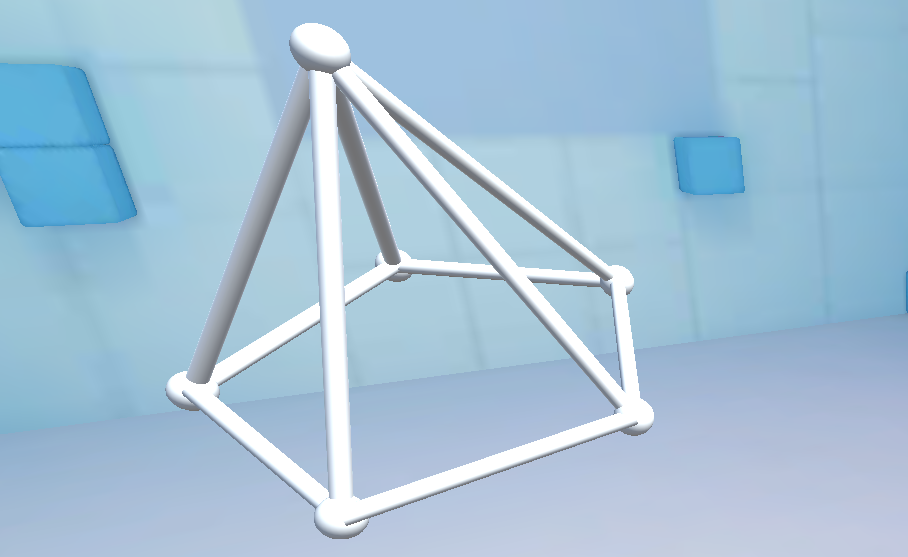
* + 1. **Delete Object / Изтриване на обект**

Всеки един обект на чертежа може да бъде индивидуално изтрит, като обаче това може да повлияе е свързаните към него други части. Поведението на всеки един обект при неговото изтриване е дефинирано в OnDestroy метода в съответния клас на обекта. Правилата при триене са:

* При изтриване на точка се изтриват и всички линии, свързани за нея
* При изтриване на линия се изтриват и точките ѝ ако това е единствената линия към която са свързани, както и всички ъгли, свързани за нея
  + 1. **Extrude (and merge to a point) / Избутване на формa**

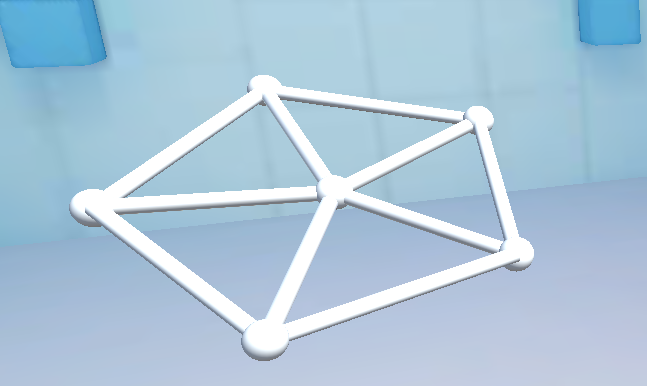
Класът Extrude дава възможността двуизмерна фигура (Фиг. 3.28) да бъде „избутана“ в триизмерното пространство (Фиг. 3.30). Има опция и тази допълнителна двуизмерна фигура да бъде компресирана в една точка, създавайки пирамида (Фиг. 3.31). Друга полза на инструмента е да бъде намерен геометричния център на дадена фигура (Фиг. 3.32).





*Фиг 3.28 Правилен петоъгълник*

*Фиг 3.30 Правилна петоъгълна призма*



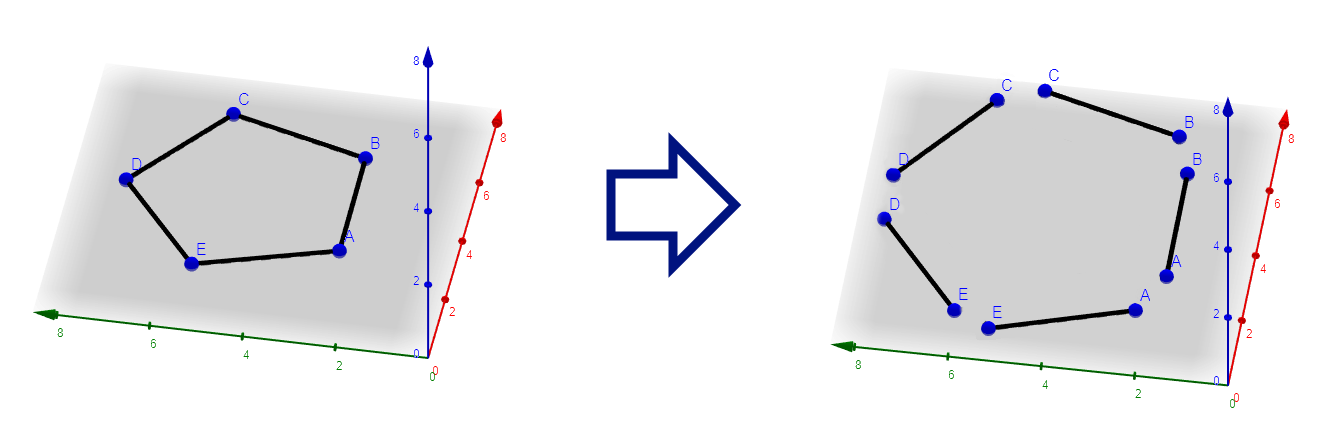
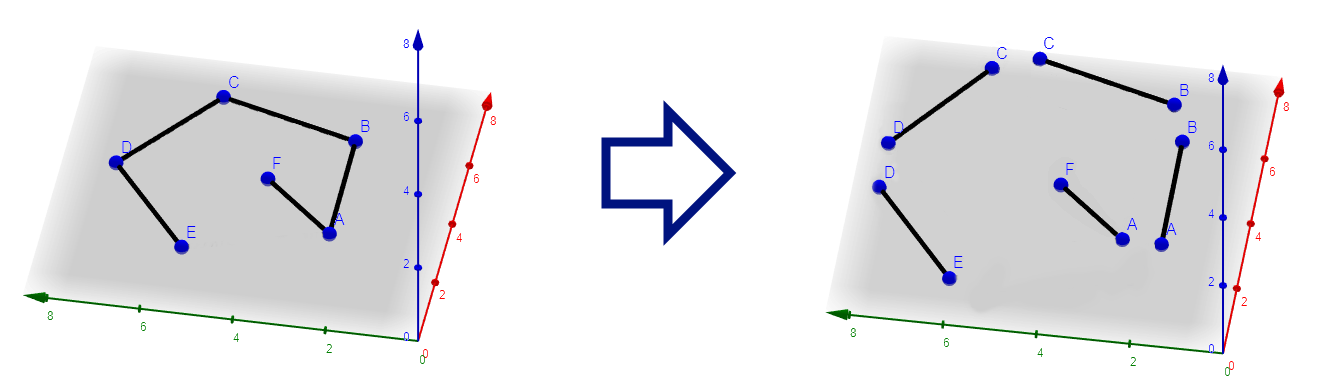
*Фиг 3.31 Правилна петоъгълна пирамида*

*Фиг 3.32 Правилен петоъгълник с център*

Самия клас съдържа в себе си четири метода:

* + - 1. **HasLoop**

След подаване на списък с линии, този метод проверява дали те образуват затворен контур. За да постигне това, първо се разделя селектирания обект на несвързани линии. След като всяка линия бива определена от две точки, тези точки трябва да се застъпват при затворения контур. Ако всяка точка се среща по два пъти – имаме затворен контур. (Фиг. 3.33) Ако ли не, селектираните линии не образуват контур (Фиг. 3.34).

След като точка се среща два пъти след разделянето, тя бива застъпена от друга в оригиналната фигура. Ако всички точки се застъпват една друга, налице е затворен контур.

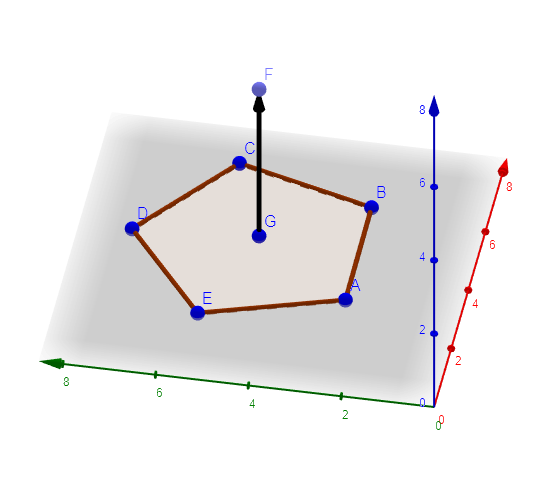
*Фиг 3.37 Разделяне на форма, която не образува затворен контур*

*Фиг 3.33 Разделяне на форма, която образува затворен контрур*

* + - 1. **GetPoints**

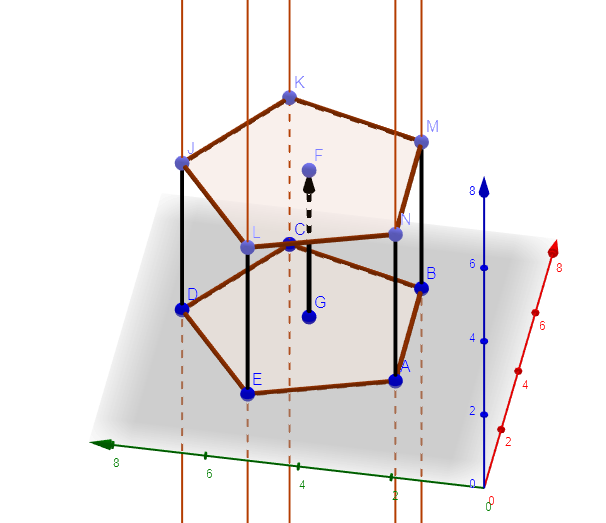
Метода GetPoints приема списък с линии и връща списък с всички точки помежду им. Може да се включи и проверка, която филтрира точките, премахвайки дупликатите. Следователно списъка, който бива връщан, може да съдърща както всички точки след разделянето на формата на линии, така и само индивидуалните такива, свързващи линиите.

* + - 1. **ChangePos**

След активиране на инструмента при правилно селектиран затворен контур, на чертежа се появява т.нар. точка-маркер. Тя се използва за насочване на вектора, пуснат от центъра на фигурата (Фиг. 3.38), като позицията ѝ може да се изменя в триизмерното пространство чрез ChangePos метода.

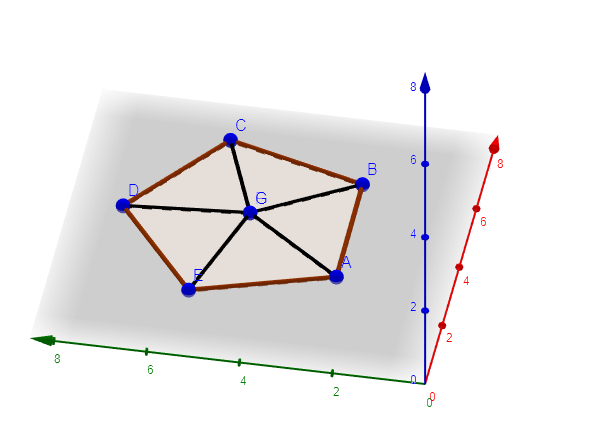
*Фиг 3.38*

* + - 1. **Confirm**

След избиране на съответно ѝ местоположение съответната фигура може да бъде преобразувана както в призма, така и в пирамида. За да бъде „избутана“ формата в пространството се създава копие на фигурата, точките на която се транслират по посока вектора с неговата дължина (Фиг. 3.39).

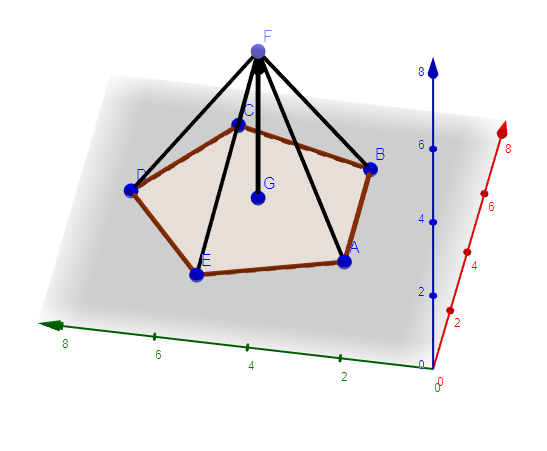
*Фиг 3.39*

Ако бъде избрано преобразуването в пирамида, всяка точка от фигурата бива свързана с точката-маркер, която се преобразува във връх (Фиг. 3.40). Има и възможност, когато дължината на вектора е нула и точката-маркер съвпада с геометричния център на фигурата да бъде построена пирамида с височина нула. По този начин може да бъде открит центъра на определена форма (Фиг. 3.41).

****

*Фиг 3.41*

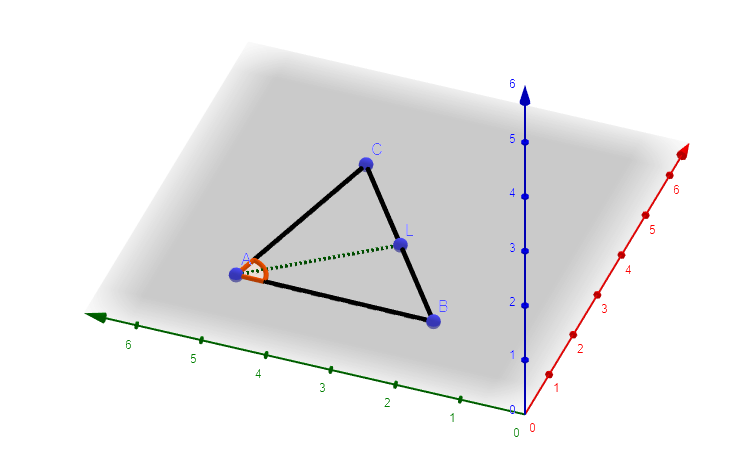
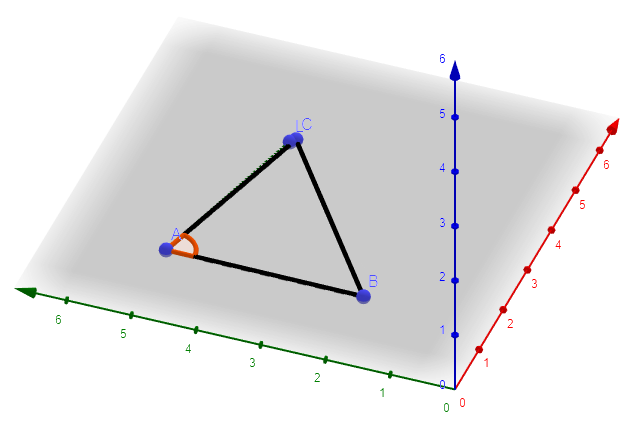
*Фиг 3.40*

****

* + 1. **Move Object / Местене на обект**
    2. **Select/Deselect All**

За селектиране/деселектиране на всички обекти на чертежа се използва класът SelectOrDeselectAll. Ако е избрано селектиране, се итерира през всеки обект на чертежа и ако не е селектиран, той бива селектиран. Иначе ако е избрано деселектиране итерира се през всеки селектиран обект и той бива деселектиран.

* + 1. **Create Bisector**

Приложението позволява построяването на ъглополовяща на даден ъгъл. При селектиране на ъгъл, неговата ъглополовяща бива спусната от върха му. За да се постигне това първо се създава нова точка на позицията на някоя от крайните точки на ъгъла. (Фиг Х.ХХ)

*Фиг Х.ХХ*

*Фиг Х.ХХ*

След това се изчислява с колко точката трябва да се транслира по правата CB за да застане в позиция, където свързана с върха на ъгъла би образувала ъглополовяща. (Фиг. Х.ХХ)

Изчислението става по следния начин:

След като CL е ъглополовяща, следва следната теорема:

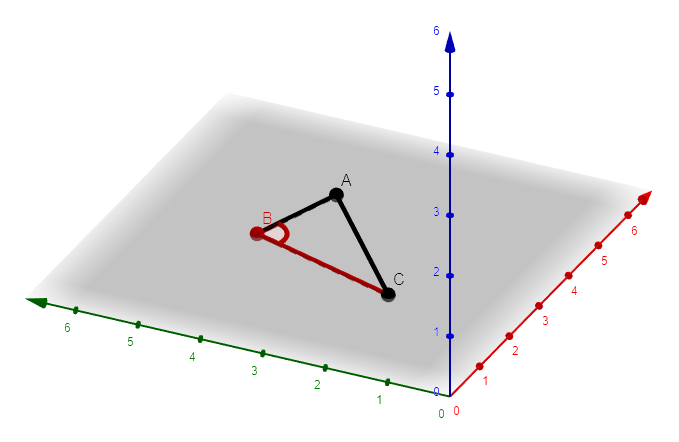
От чертежа може да заключим че , или ако прехвърлим получаваме че . След заместване в горната формула вече имаме че

От тук след умножение на кръст следва че ,

но след прехвърляне получаваме . Изнасяме общия множител пред скоби и изразяваме CL: . По този начин получаваме дължината с която да транслираме точката по правата CB.

Остава само самата ъглополовяща да бъде построена чрез свързването на новосъздадената точка и върха на ъгъла.

* + 1. **Create Circle**

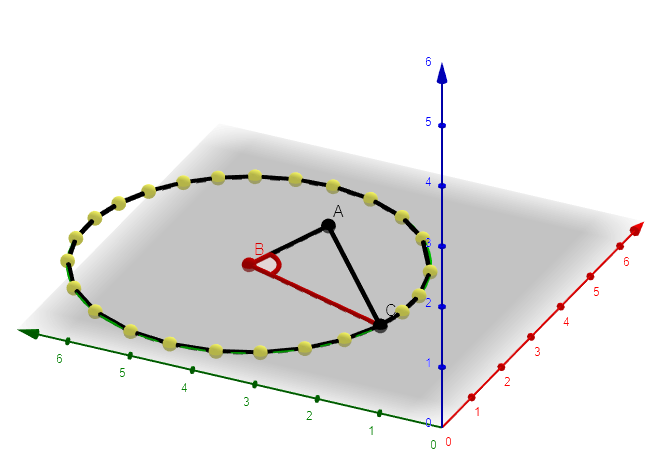
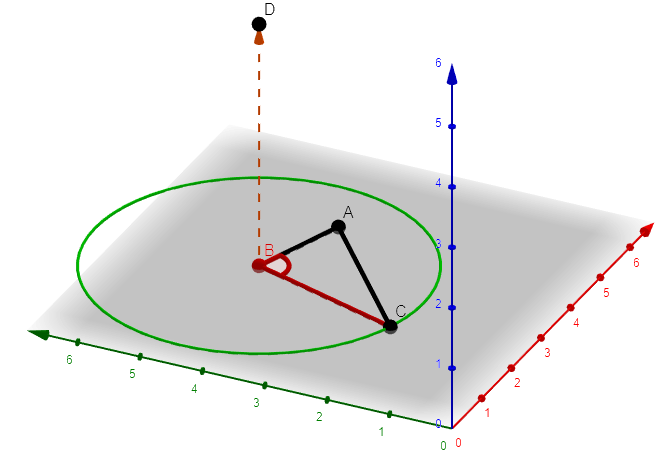
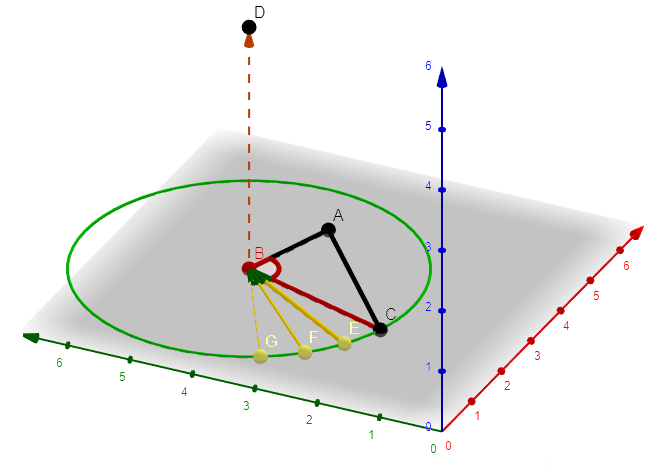
При селектиране на точка, линия и ъгъл една единствена окръжност бива дефинирана. Тази окръжност бива създадена с център селектираната точка, радиус – селектираната права и ротация – избрания ъгъл.

*Фиг 3.42 Представяне на селектираните обекти чрез червен цвят*

След като определените елементи биват селектирани (Фиг. 3.42) биват изчислени следните неща: векторът , който е перпендикулярен на равнината чрез векторното произведение на и ; оптималния брой точки, които да описват окръжността чрез следната формула:

както и ъгъла между всеки две точки на окръжността чрез делението

След като всичко това бива изчислено (Фиг. 3.43) на позицията на т.C се създава първата нова точка от отръжността. Тази новосъздадена точка бива завъртяна около вектора с ъгъл, равен на , като индексът започва от 1 и се увеличава с единица за всяка следваща точка (Фиг. 3.44). Накрая всички създадени точки биват свързани помежду си чрез линии. (Фиг. 3.45)

****

*Фиг 3.45*

*Фиг 3.44*

*Фиг 3.43*

**ЧЕТВЪРТА ГЛАВА**

Ръководство на потребителя

1. **Инсталиране**

Инсталирането може да се осъществи по два начина: Първият начин е през Google Play приложението или уебсайта, а втория – чрез файл с разширение apk.

* 1. **През Google Play**

След влизане в Google Play приложението, което е инсталирано на всеки телефон с операционна система Android, се търси приложението по име или по производител. След откриването му може да бъде видяно описание на приложението, снимки и последни актуализации.

При натискане на бутона за инсталиране от страницата, приложението ще започне да се сваля, а след това и ще бъде инсталирано автоматично от системата.

Ако инсталирането трябва да бъде осъществено от уеб портала на Google Play, то е необходимо да се намери страницата на приложението по същия начин и да се натисне бутона за инсталация (трябва да е влязъл регистриран профил). Веднага щом телефонът получи достъп до интернет, приложението ще започне да се сваля и ще бъде инсталирано отново от системата.

* 1. **Чрез apk файл**

За да бъде инсталирано от apk файл, най-напред е необходимо да бъде разрешено инсталирането на неоторизирани приложения от:

Settings > Security & Privacy > Install unknown apps

След това apk файла трябва да бъде качен на мобилното устройство посредством USB, Bluetooth или по друг начин. Откриването на прехвърления файл става посредством файлов мениджър като след стартирането му системата ще попита за потвърждение дали приложението да бъде инсталирано. След преглед на исканите права и разрешаване на инсталацията приложението ще бъде инсталирано, като след това иконата му ще се появи в менюто.

1. **Стартиране**
2. **Използване**
   1. **Select**
   2. **Ц.**
   3. **Етц.**
3. **Деинсталиране**

Премахването на приложението става по стандартния за Android платформата начин – от менюто „Приложения“ в настройките на устройството.

1. **Поддръжка**
   1. **Актуализации**

Актуализациите (или ъпдейтите) се предоставят от Google Play приложението в момента в който такава е налична и телефона има интернет връзка

* 1. **Докладване за грешки**

Предоставя се от Android платформата. Когато приложението неочаквано се изключи или блокира поради някаква причина, ще се появи диалогов прозорец, в който може да се избере да се докладва грешката.

**ЗАКЛЮЧЕНИЕ**

**ИЗПОЛЗВАНА ЛИТЕРАТУРА**

**СЪДЪРЖАНИЕ**