**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: Екшън игра от първо лице във Виртуална Реалност

Дипломант: Научен ръководител:

*Даниел Янев Виктор Кетипов*

СОФИЯ

2019

Увод

Продуктът представлява игра от първо лице в Виртуална Реалност. Целевата аудитория на продукта е геймъри, които вече притежават комплект за Виртуална Реалност със система за контрол на движение. Такива система в момента на пазара са:

* **Oculus Rift** (Избран за разработка)
* HTC Vive

Целта на играта е защита на обект (в случая бариера) от врагове. Играта включва 5 вълни от врагове, 5 различни врагове, xsad различни оръжия, система за купуване на оръжия/ъпгрейди. Играчът може да държи едновременно максимум 2 оръжия (по едно във всяка ръка).Играта приключва или когато паднат и двете прегради от едната страна или когато играчът успешно победи всички вълни от врагове.

Основното движение се установява чрез VR комфортен режим (VR Comfort Mode). Отличаващ се с един вид микро-телепортация, VR комфортният режим забранява плавния таван, който може да направите с десния стик, и го заменя със завъртане на точно определен интервал от градуси.

Локацията на базата на погледа - т.е. „изглежда там и отиде там“ - също е популярен избор сред разработчиците, които включват VR комфортен режим в игрите от първо лице, но е по-малко естествено от завъртания на точни интервали.

ПЪРВА ГЛАВА

Технологии за реализация на дипломния проект

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



*Фиг 1.1 Unreal Engine лого*

Unreal Engine е един от най-разпространените публични игрови двигатели („game engines“) на пазара. Поддръжката на всички масови платформи заедно с безкомпромисното качество и вече 21 години разработка показват, защо това е един от най-популярните и често използвани игрови двигатели. Unreal Engine е изцяло написан на C++, като целия код е публично достъпен в Github.

Unreal Engine се грижи почти изцяло за рендерирането и менажирането на памет, създавайки абстракция за програмиста.

Няма преки пътища към създаването на потапящи се преживявания, които са правдоподобни за човешкия ум. VR изисква сложни сцени, представени при много високи стойности на кадрите. Тъй като Unreal Engine е предназначен за взискателни приложения като AAA игри, създаване на филми и фотореалистични сцени, той отговаря на тези изисквания и осигурява солидна основа за изграждане на продукти за всички VR платформи - от компютри през конзоли до мобилни

-Wikipedia

Главните предимства при използване на игрови двигател са, че те се грижат за:

* Рендериране
* Управление на памет
* Засичане на колизии и още много.

Като бонус Unreal Engine предоставя и невероятно лесно интегриране на Виртуална Реалност.

* + 1. **C++**

*Фиг 1.2 C++ лого*

**„***C++ или CPP („Си Плюс Плюс“) е език за програмиране с общо предназначение. Той има императивни, обектно-ориентирани и общи програмируеми функции, като същевременно осигурява възможности за манипулиране на паметта на ниско ниво.“* – Wikipedia

Unreal Engine имплементира собствен „Garbage Collector“ и ” Unreal Property System (Reflection)“ система базирана изцяло на макрота. Две функции, нужни на един модерен игрови двигател, които C++ като език известен с максимална производителност, ефективност и гъвкавост не предлага.

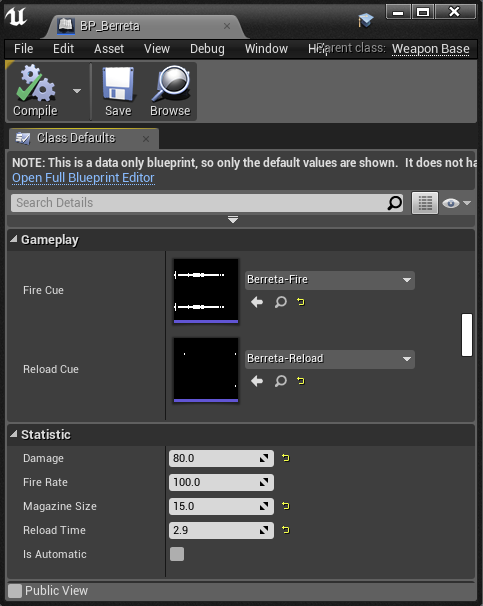
Системата „Garbage Collector” изцяло се грижи за управлението на памет като следи, кои обекти се използват и кои вече не се. “Modern C++” имплементира няколко „умни поинтери“ („Smart pointers”), но Unreal Engine скрива и това от програмистите като менажира цялата памет.

“Reflection” системата дава способността на една програма да се изследва по време на изпълнение. Това е изключително полезно и е основополагаща технология на Unreal Engine, много системи като детайлни панели в редактора, сериализация, „Garbage Collection”, репликация на мрежата и комуникация „Blueprint“ / C ++, разчитат на тази система. C ++ обаче не поддържа под никаква форма „Reflection” система, така че Unreal има своя собствена система за събиране, обработване и манипулиране на информация за C ++ класове, структури, функции, член променливи и енумерации.

„Reflection” системата не е автоматична. За да бъде „видим“ за нея тип или свойство на един клас трябва да се анотира. „Unreal Header Tool“ ще събере тази информация, когато компилирате проекта.



*Фиг 1.3 Демонстрация на „Unreal Property System (Reflection)“ в код*



*Фиг 1.4 Демонстрация на „Unreal Property System (Reflection)“ в редактора*

* + 1. **Oculus API**

Всички версии на Unreal 4.10 и по-късно включват вградена поддръжка за Oculus Rift и Samsung Gear VR, включително автоматично стереоскопично визуализиране и проследяване. Поддръжката на Oculus Go е включена в Unreal Engine 4.19 и по-късно. Unreal се грижи за цялата комуникация между софтуера и “Oculus API”, което управлява, както очилата за виртуална реалност, така и контролерите за движение.

* 1. **Хардуер**
     1. **Oculus Rift**

*Фиг 1.5 Oculus Rift + Touch bungle*

Oculus Rift е платформа за виртуална реалност, разработен от купената от Facebook компания Oculus. Rift има два Pentile OLED дисплея, 1080 × 1200 резолюция всеки, 90 Hz честота на опресняване и 110 ° зрително поле. Очилата за Виртуална Реалност (HMD – Head Mounted Display) притежава и ротационно и позиционно проследяване, както и вградени слушалки, които осигуряват 3D аудио ефект.

* + 1. **Touch bundle**

*Фиг 1.6 Oculus Touch bungle*

Системата за контрол на движението на Oculus Rift е известна като Oculus Touch. Състои се от двойка преносими устройства, по една за всяка ръка, всяка от които съдържа аналогова пръчка, три бутона и два тригера (един често използван за хващане, а другият за стрелба). Контролерите са напълно проследени в 3D пространството от системата “Constellation”, така че те могат да бъдат представени във виртуалната среда, и всеки контролер разполага със система за откриване на жестове с пръсти, които потребителят може да прави, докато ги държи. Всеки контролер използва една AA батерия.

Контролерите представляват единственият (като изключим движението на HMD) начин на потребителя да взаимодейства със софтуера.

ВТОРА ГЛАВА

Проектиране на структурата на игрови приложения за Виртуална Реалност

1. **Функционални изисквания към приложението**
   1. **Изисквания към изграждането на логиката**

Трябва да бъде изградена основна 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 по „Good practice” конвенцията на Unreal Engine.

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

Потребителският интерфейс на приложението трябва да бъде максимално опростен и разбираем. Цялата навигация ще трябва да се осъществява само чрез реални 3D обекти в играта, затова всички менюта и бутони трябва да са с достатъчно голям размер че да не бъде изпитвано затруднение при използването и разчитането им.

* 1. **Функционалност**

Приложението трябва да има минимум 5 различни вълни от врагове, 5 различни врагове и система за купуване на оръжия.

* 1. **Производителност**

Приложението не трябва да забива или замръзва без причина и е желателно да поддържа средно 90 кадъра в секунда като минимума не пада под 60. Трябва да се използва метод за движение, който не причинява гадене и замайване при продължително използване на Виртуалната Реалност. Моменти на насичане, прекъсване или други дефекти не трябва да бъдат забелязвани.

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

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

* Спестява много време и усилия като се грижи за паметта, рендериране и дава много абстракции и вече готови и имплементирани класове, както и инструменти за работа с 3D модели, анимация и т.н.
* Повечето игрални двигатели са междуплатформени, както и предоставят интегрирана поддръжка за Виртуална Реалност.

Игровият двигател, който беше избран е Unreal. Причината е че е “Open Source”, инструментите му не са непознати. Това многократно ускорява процеса на работа.

* 1. **Oculus Rift платформа**

Продуктът е насочен към компютърни ентусиасти, по простата причина, че една игра във Виртуална Реалност е много по-изискваща от страна на ресурс. На пазара има много мобилни платформи за Виртуална Реалност, но нито една не успява да предостави достатъчно ресурси, за да е приятно и максимално близко до реалността изживяването.

* 1.  **GitHub система за контрол и управление на версиите**

*Фиг 2.1 GitHub лого*

GitHub Inc. е уеб-базирана хостинг услуга за контрол на версиите чрез Git. Git система за контрол е избрана заради вече съществуващия опит с нея.

* 1. **Visual Studio 2017 текстов редактор**

*Фиг 2.2 Visual Studio лого*

Unreal Engine има вградена интеграция с Visual Studio, която улеснява компилиране, менажиране на проекта и търсене в кода на игровия двигател.

* 1.  **Blender графичен редактор**

*Фиг 2.3 Paint.net лого*

*Blender е безплатен „Open Source” 3D софтуер, който се използва за създаване на анимационни филми, визуални ефекти, изкуство, 3D печатни модели, интерактивни 3D приложения и видео игри. - Wikipedia*

ТРЕТА ГЛАВА

Програмна реализация на проекта

1. **Архитектура на приложението**
   1. **Основни класове за всеки Unreal Проект**

Архитектурата на Unreal Engine разчита на няколко основни класа за да контролира играта.

* + 1. **Game Mode („Игрови Режим“)**

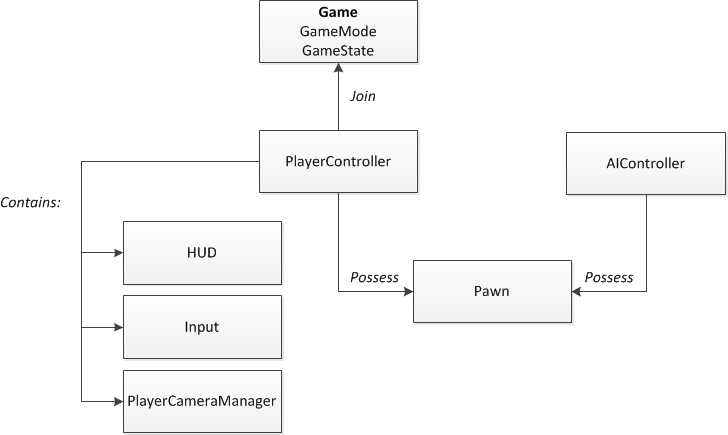
В архитектурата на Unreal игровият режим се грижи за следните неща:

* Броят на присъстващите играчи и зрители, както и максималният брой играчи и зрители.
* Как играчите влизат в играта, които могат да включват правила за избор на места за инстанциране (spawn) и друго поведение.
* Дали играта може да бъде поставена на пауза и правилата на паузата.
* Преходи между нива, включително дали играта трябва да започне кинематографичен монтаж.

В случая игровият режим се грижи и за състоянието на играта, ресурси и стигането на информация за тях до играча и отговаря аз инстанцирането и позиционирането на враговете в сцената.

* + 1. **Player Controller („Контролер на играча“)**

Всяка пионка (“Pawn“, обект който може да бъде контролиран) се контролира или от контролер за играч (Player Controller) или от контролер с изкуствен интелект (AI Controller).



*Фиг 1.1 Структура на архитектурата на Unreal за управление на игра*

Тъй като играта се играе от един играч, имаме само една инстанция на Player Controller, който по подразбиране се контролира от играча и е връзката между играча и обекта, който той контролира.

* + 1. **Default Player Pawn (Обекта, който контролира играча по подразбиране)**

Този обект за грижи за приемане на информацията от контролера и имплементиране на движение, стрелба, анимации, взаимодействие със средата (валидно за всеки един Pawn). В случая той създава и пази референция към шлема за виртуална реалност и двете устройства за проследяване на ръцете (Motion Controllers).

* 1. **Разделение на класовете**

Всеки тип враг, оръжие, анимационнен контролер (Animation Controller)/реф/ наследява съответно базов клас, който имплементира общи функционалности.

|  |  |
| --- | --- |
| Функция | Съответстващ клас |
| Базов клас за враг | AZombieBase |
| Базов клас за оръжия | AWeaponBase |
| Базов клас за анимации | UZombieBaseAnimationInstance |
| Базов клас за “снаряд” | AProjectileBase |
| Базов клас за UI с един параметър | UOneParamWidget |

* 1. **Помощни класове** 
     1. **UInteractableComponent**

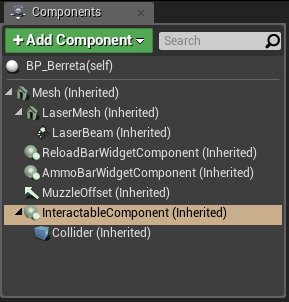
Чрез този клас компонент всеки обект може да взаимодейства с друг в света. Предоставя интерфейси както за прости взаимодействие (като включване, изключване),така и за използване на предмети като оръжия с множество контроли.

UInteractableComponent класа разчита на така наречените Delegates(делегати), които Unreal Engine поддържа.

Делегатите позволяват да бъдат извиквани член функции на C ++ класове по общ, но безвреден за типа начин (type-safe). Използвайки делегати, може динамично да се свържат с член функция на произволен обект, след това да бъдат извикани функции на обекта, през делегата, без да е известен вида на обекта.

Копирането на делегати е напълно безопасно. Те могат да бъдат предавани по стойност, но това обикновено не се препоръчва, тъй като те заделят памет в heap.

Делегатите биват няколко вида - Single-cast, Multi-cast, Events и Dynamic (UObject, serializable).

Всеки обект, който трябва да може да бъде използван или взаимодействан с, трябва да притежава UInteractableComponent в йерархията си от компоненти и да е инициализирал желаните методи.

*Фиг 1.2 InteractableComponent в йерархията на компонентите на обект*

Комуникацията се осъществява като при създаване на обекта на InteractableComponent-а се подават референции към вече създадени в обекта делегати.



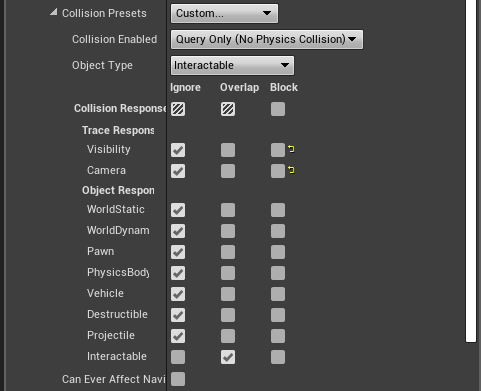
*Фиг 1.3 Дефиниция на делегатите*



*Фиг 1.4 Делегати, които се подават по референция на InteractableComponent*



*Фиг 1.5 Свързване на член функции към съответните делегати и предаване на референция към тях на InteractableComponent*

 UInteractableComponent има дъщерен компонент UBoxComponent, чиято колизия се използва за разпознаване и управляване на компонента. Колизията е от тип “Interactable” и „отговаря“ само при припокриване с друга колизия от тип „Interactable” (Фиг. 1.6)

*Фиг 1.6 Свързване на член функции към съответните делегати и предаване на референция към тях на InteractableComponent*

UInteractableComponent не имплементира никаква логика. Той е предназначен да бъде контролиран изцяло от външен обект, чрез следните публични методи (всеки извиква метода ExecuteIfBound() на съответния делегат):

|  |  |
| --- | --- |
| * void BeginUse() | * void EndUse() |
| * void BeginAction() | * void EndAction() |
| * void BeginGrab(USceneComponent \* AttachActor) | * void EndGrab() |
| * void Select() | * void Deselect() |

* 1. **Главни класове на приложението**
     1. **AWeaponBase**

Основният клас, който всички оръжия наследяват чрез “Blueprint”. Имплементира стрелба (както автоматична така и ръчна), зареждане, потребителски интерфейс и предоставя интерфейс за контрол през InteractableComponent. (Виж 1.3.1)

* + - 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 поможни 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).



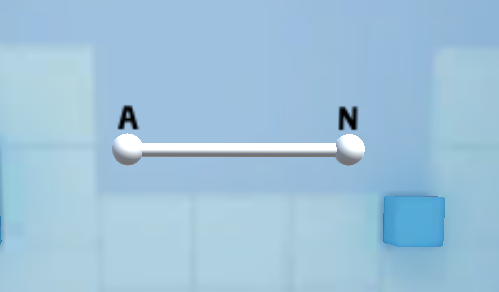
*Фиг 3.3*

*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)

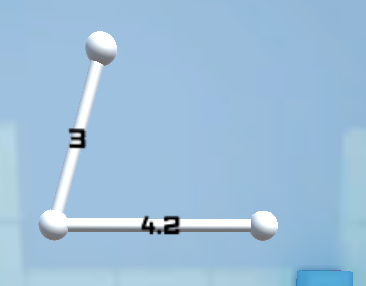
* + - 1. **ConnectAngle, DisconnectAngle**

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**
  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*



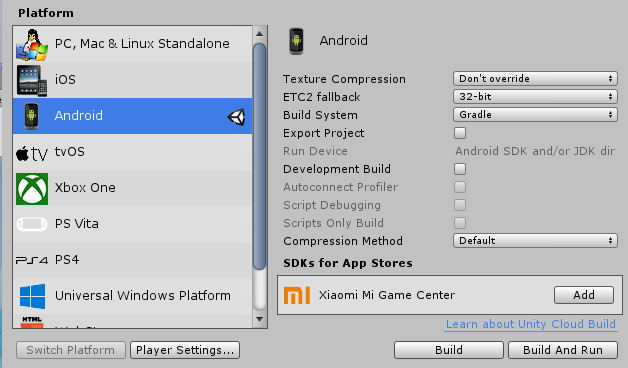
*Фиг 3.3 Курсора, насочен към обект*

* + 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.

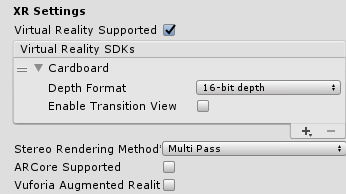


*Фиг 3.4 Менюто за избор на платформа*

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**

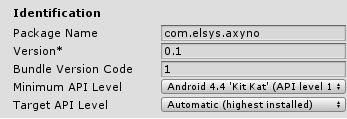
След импортиране на пакета, предназначен за разработване на приложения за Google Cardboard, от менюто за настройки на разширената реалност трябва да бъде селектирано поддържането на виртуална реалност (Virtual Reality Supported), а след това да бъде избрано „Cardboard“ като пакет. Менюто се намира на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

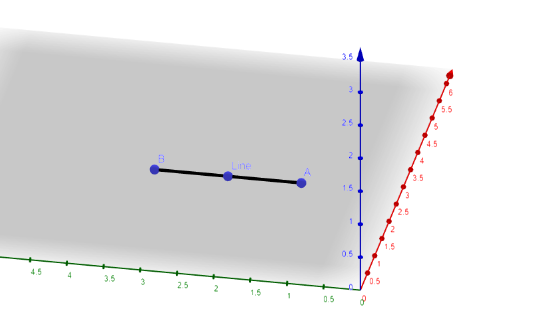
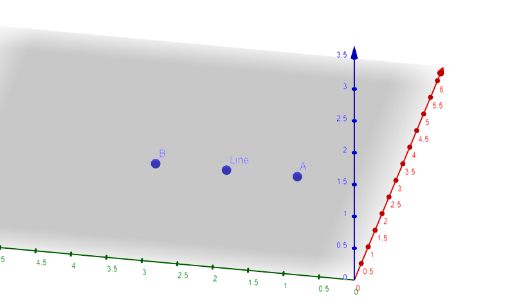
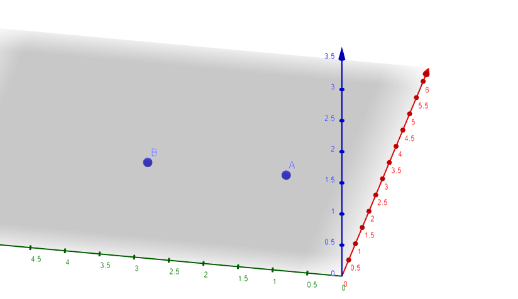
1. **Графика на приложението**
   1. **Потребителски интерфейс**
   2. **Графично представяне на елементите (shaders n such)**
2. **Реализация на инструментите и действията**
   1. **On-screen buttons**
      1. **Отваряне на меню**
      2. **Избор на инструмент**
      3. **Завъртане на чертежа**
      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*

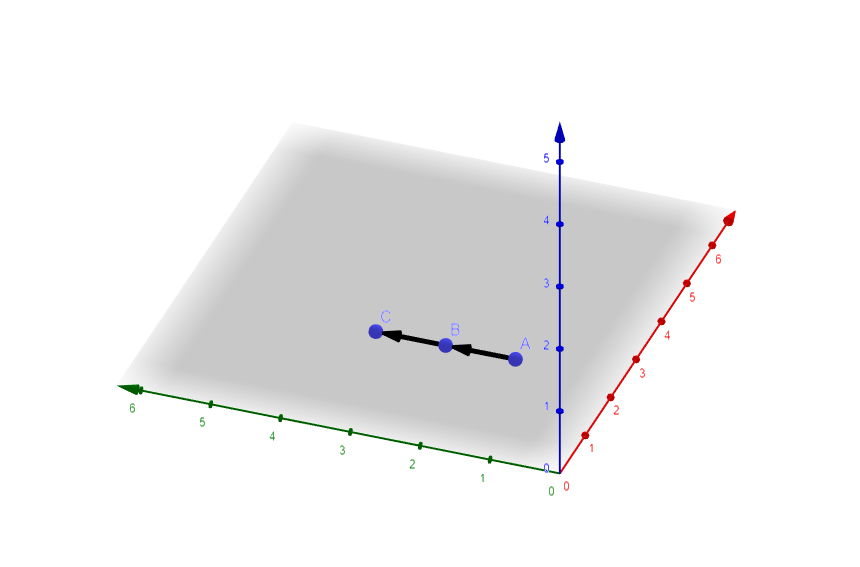
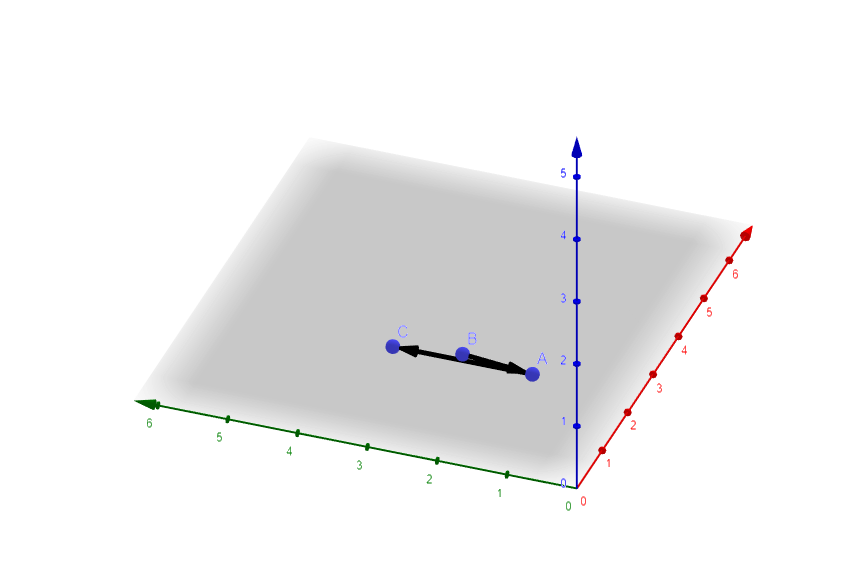
*Фиг 3.8*

* + - 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).

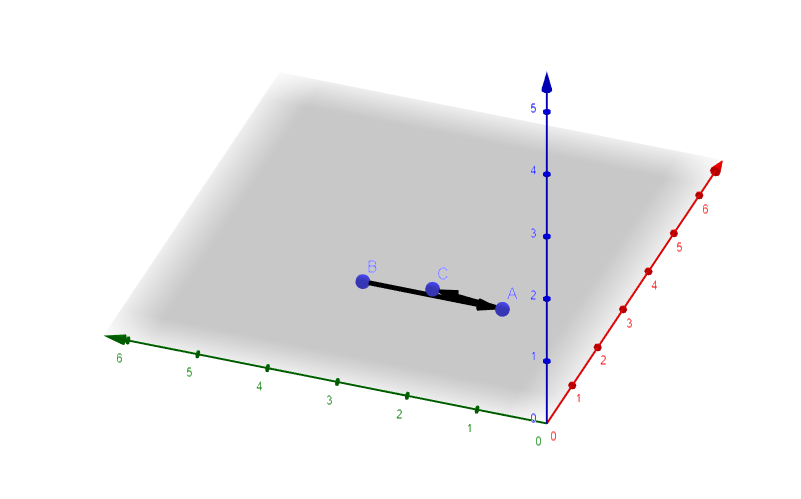
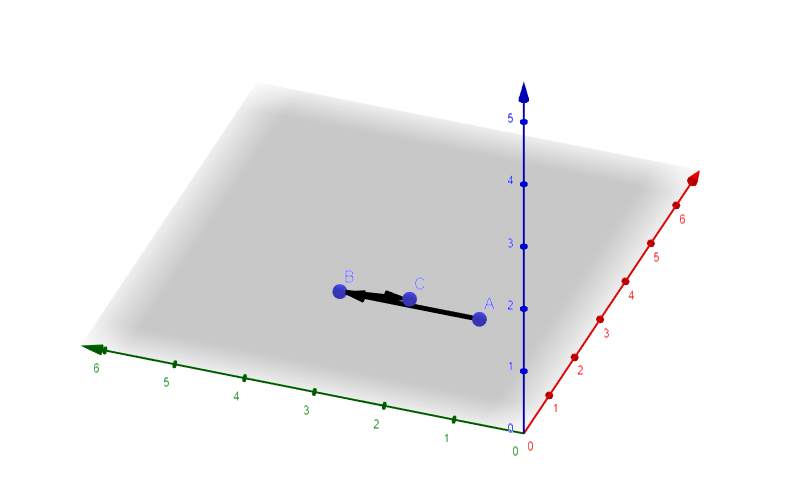
Например ако точката не лежи между другите две:

*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:

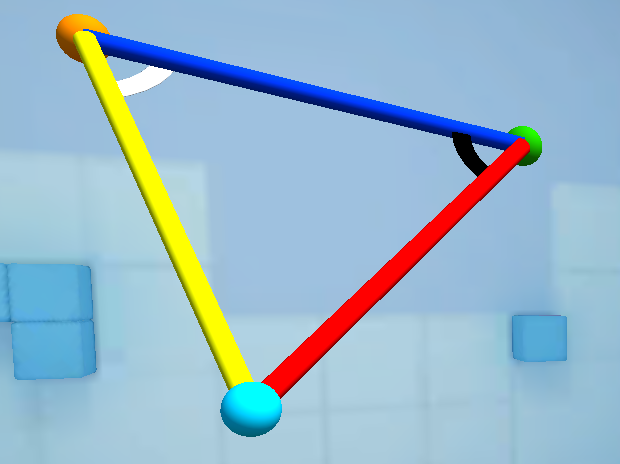


*Фиг 3.13*

*Фиг 3.12*

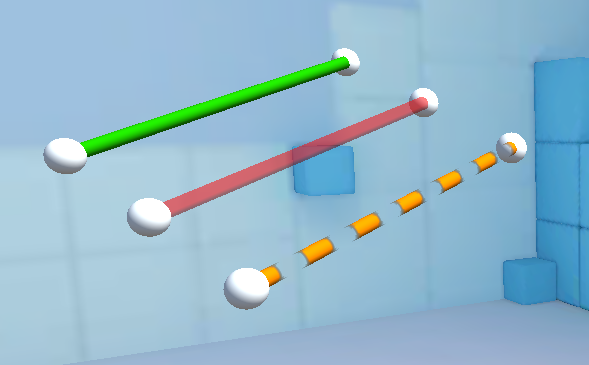
Векторите и ще бъдат с противоположни посоки (Фиг. 3.12), както и и (Фиг. 3.13). Следователно и двете скаларни произведения ще бъдат -1.

* + 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.

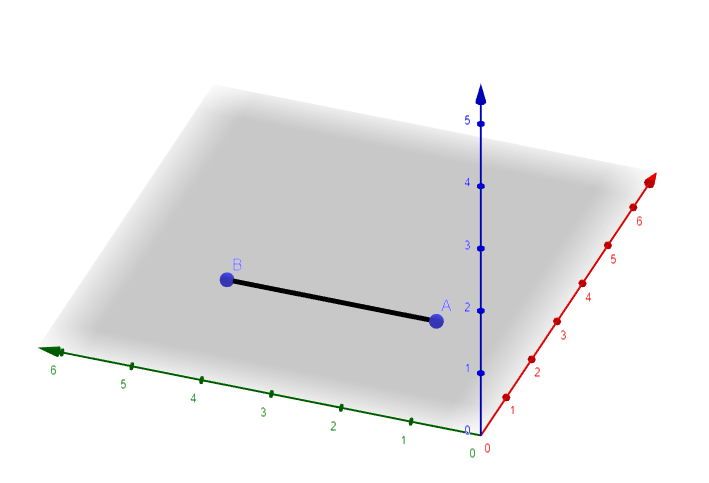
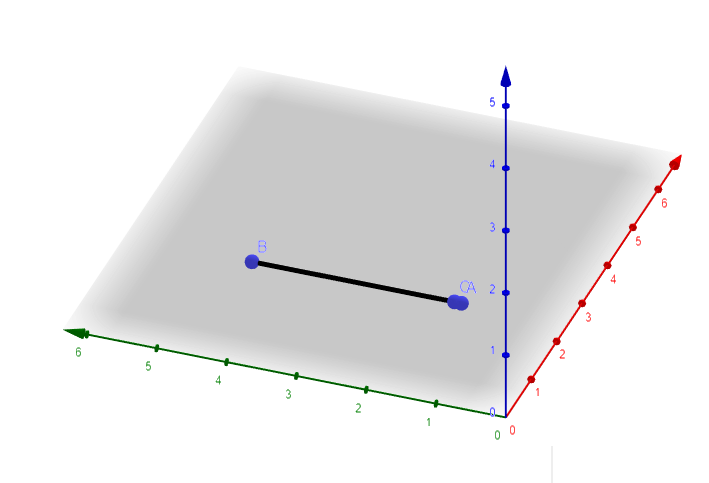
*Фиг 3.14 Триъгълник с различно оцветени компоненти*

* + 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

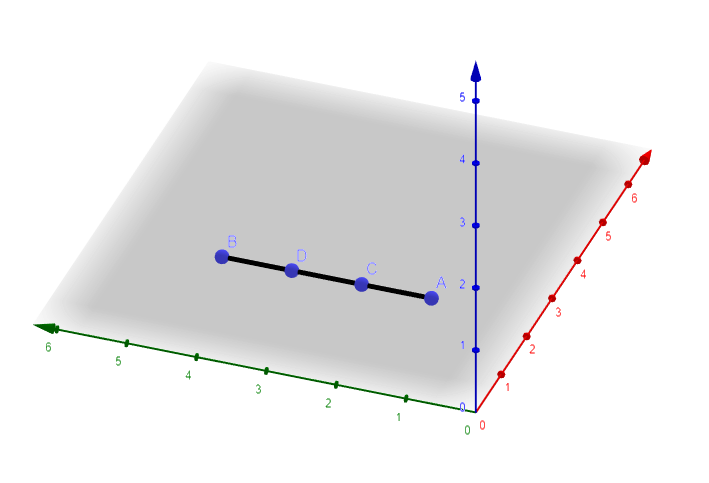
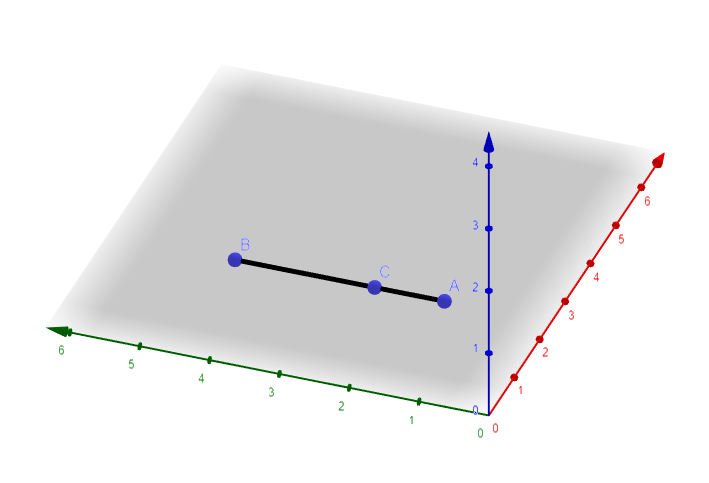
*Фиг 3.15 Линии от различен тип и цвят*

* + 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).

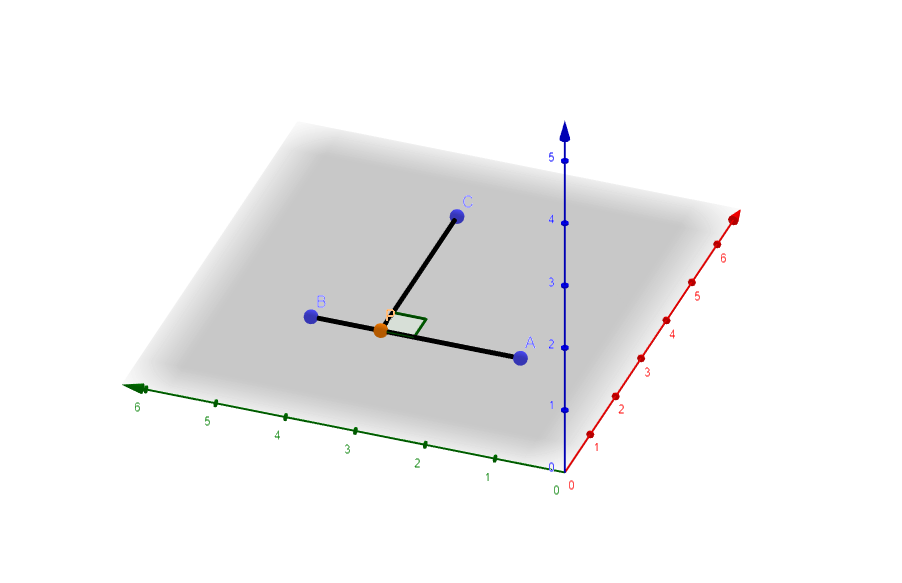
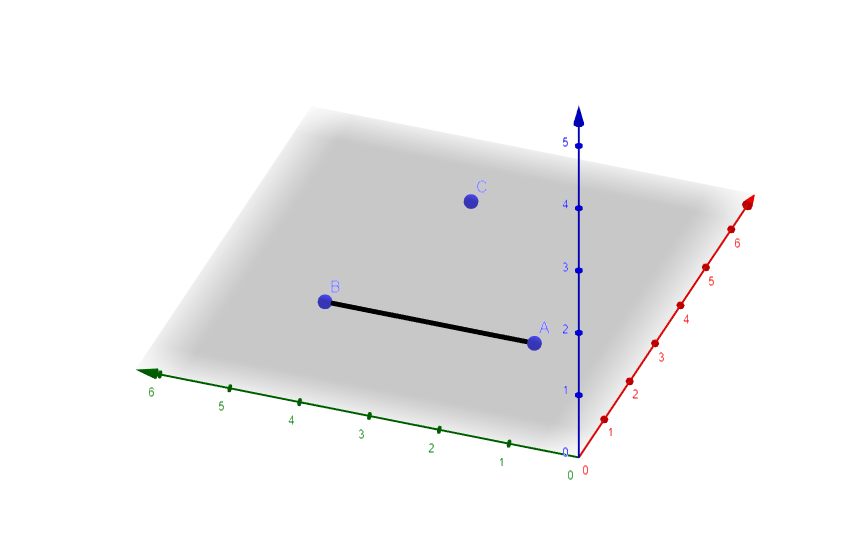
*Фиг 3.18*

*Фиг 3.19*

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

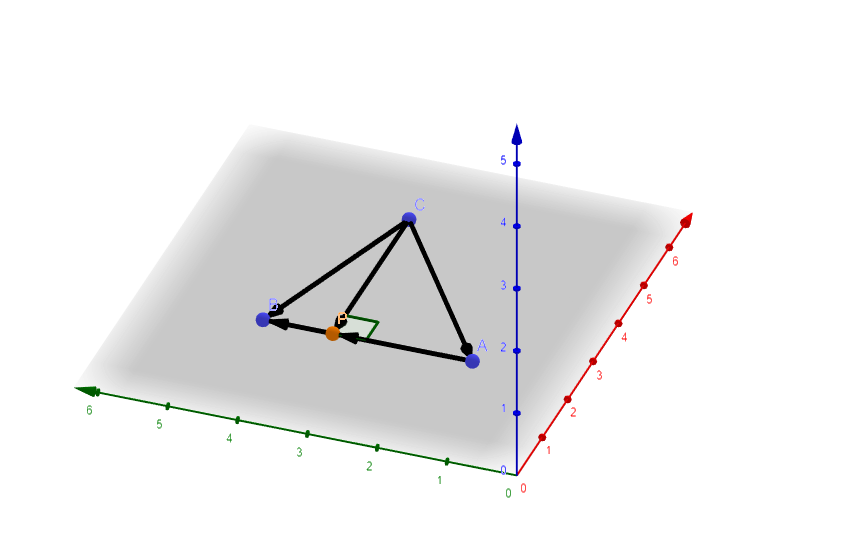
* + 1. **Build Perpendicular / Строене на перпендикуляр**

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:

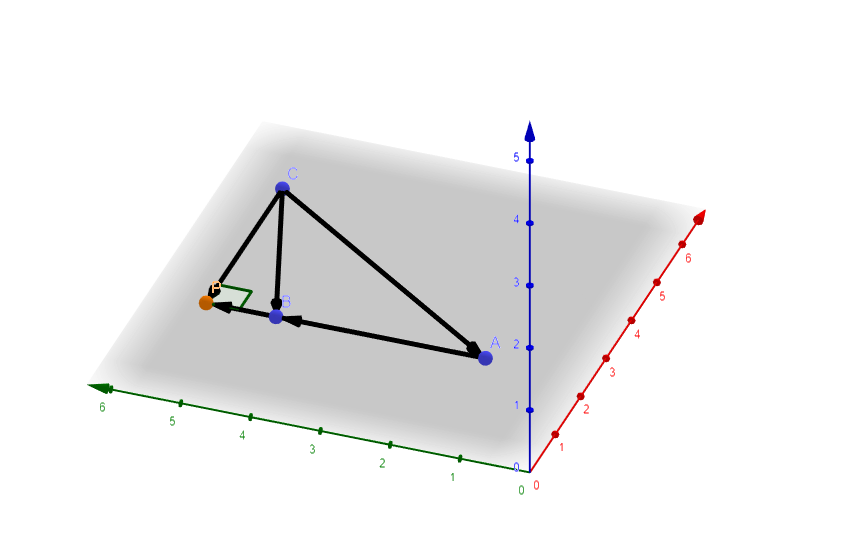
След съкращаване на съответните координати на точка C от двете страни, финалната формула става

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**

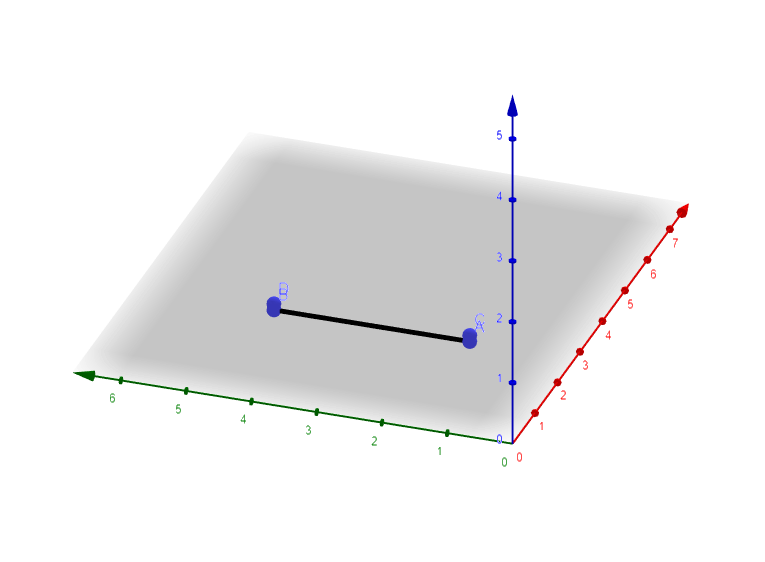
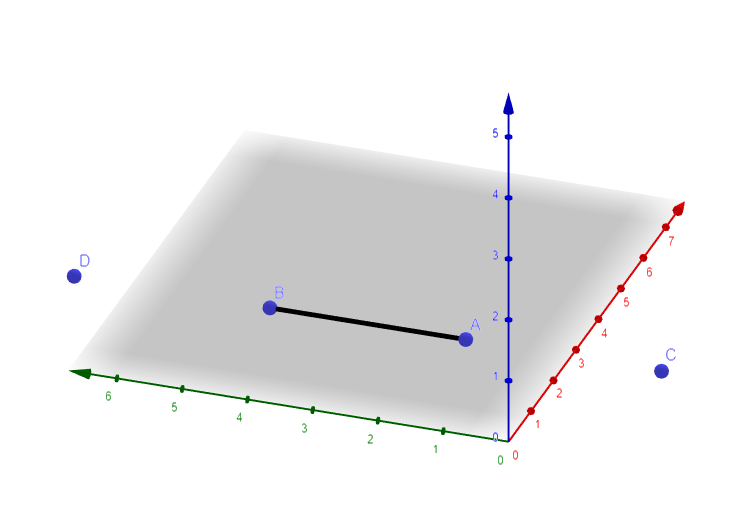
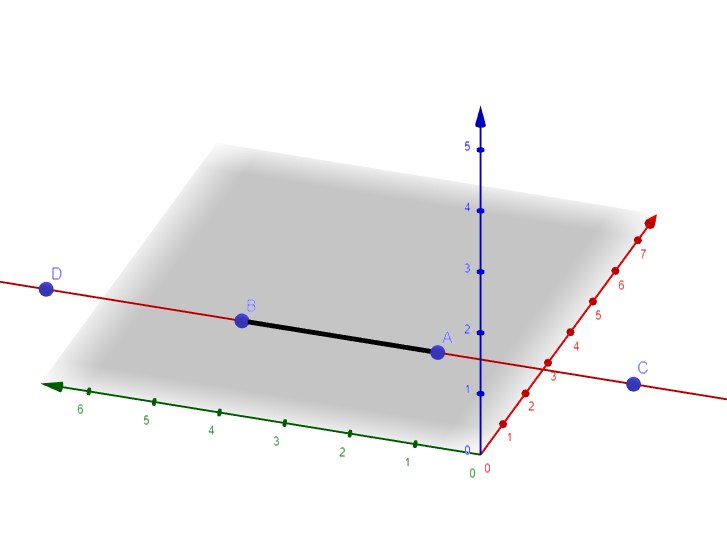
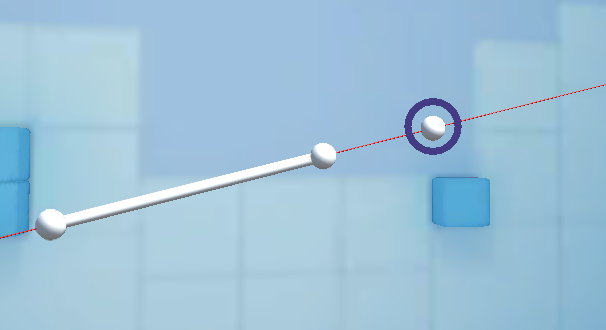
За създаването на обекти от типа ъгъл, се грижи класът CreateAngle. При селектиране на две линии от чертежа, се създава обект от типа ъгъл. След това новосъздадения ъгъл бива свързан към линиите и му бива зададена определената форма и позиция чрез UpdateAngle метода му.

*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 3.25*

*Фиг 3.24*

*Фиг 3.26*

*Фиг 3.27*

* + - 1. **LineHover**

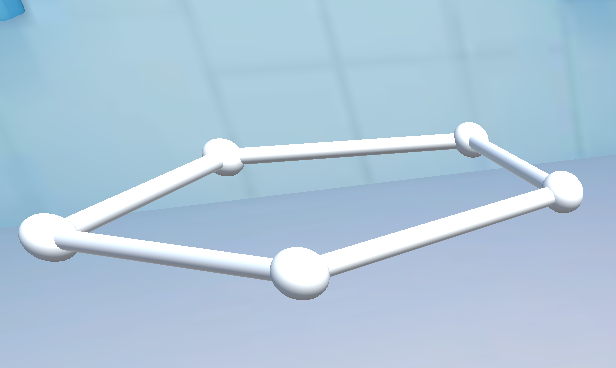
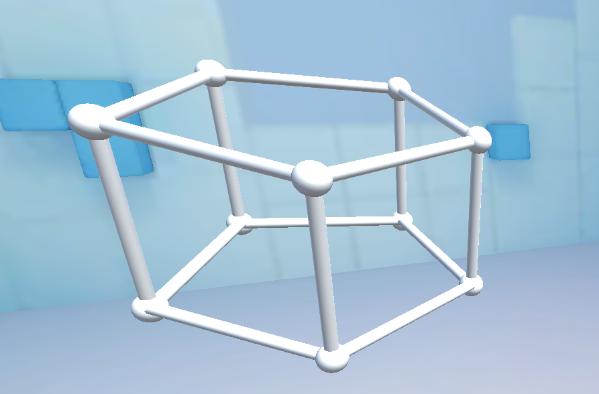
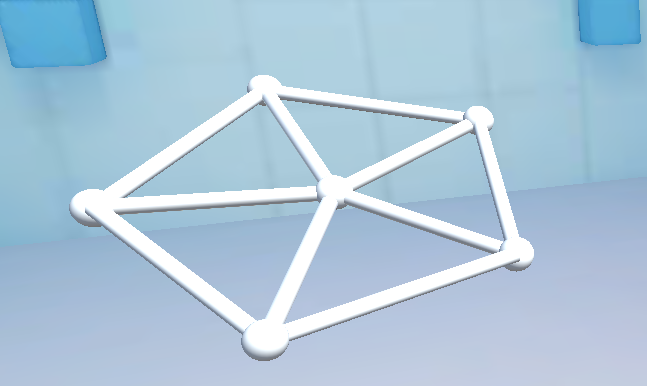
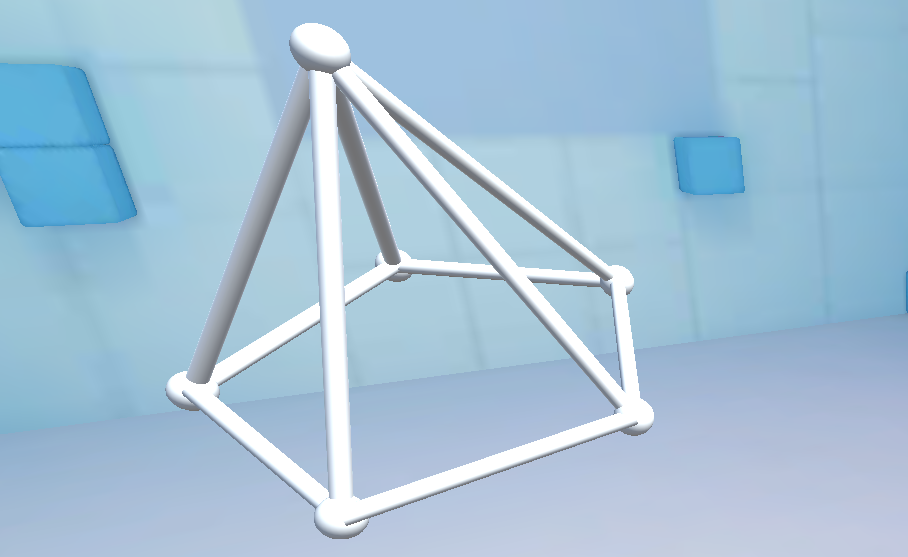
Този клас се ползва само и единствено от спомагателната линия. При местене на курсора по правата може предварително да се види позицията на бъдещата точка (Фиг. 3.27). След потвърждение от страна на потребителя, новата точка бива създадена и ако позицията ѝ е извън правата бива свързана с по-близката 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*

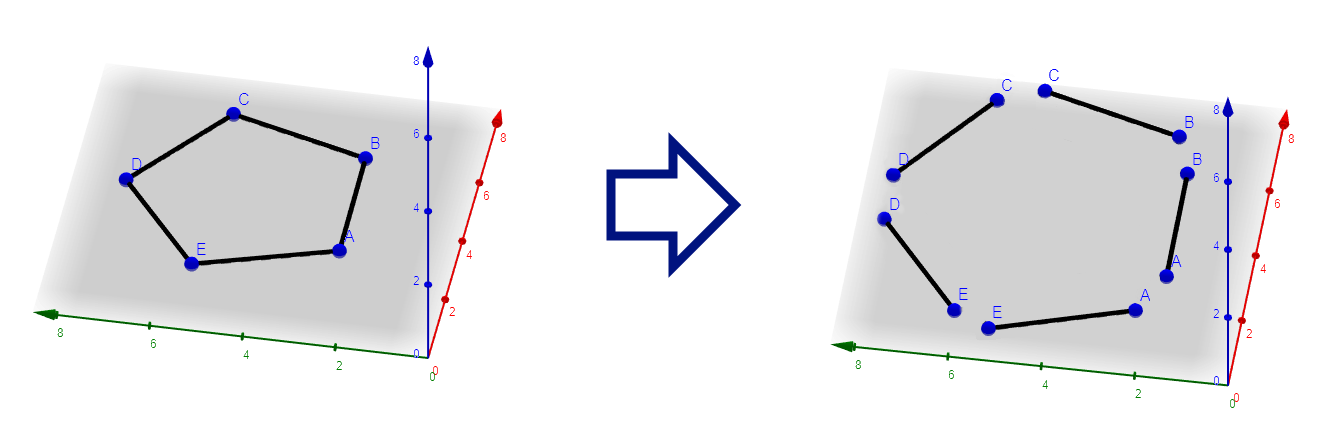
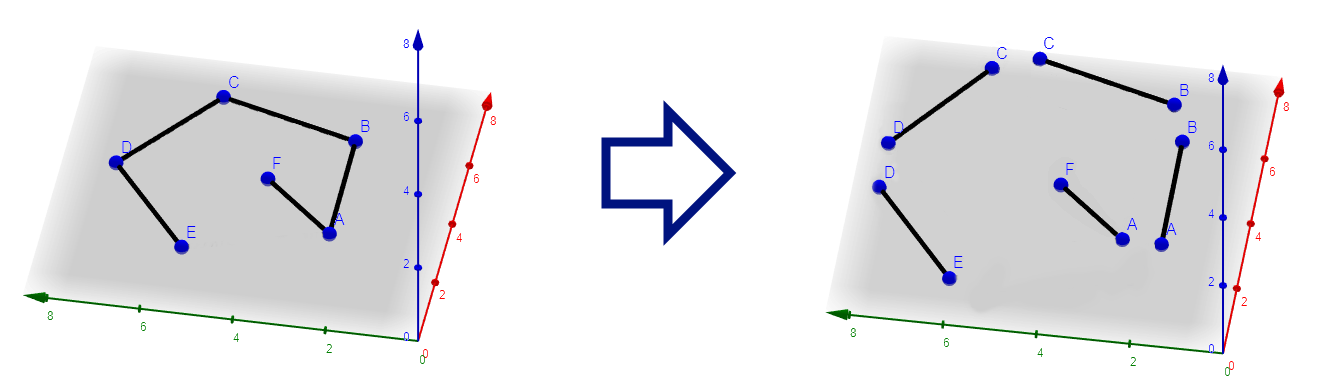
*Фиг 3.31 Правилна петоъгълна пирамида*

*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.

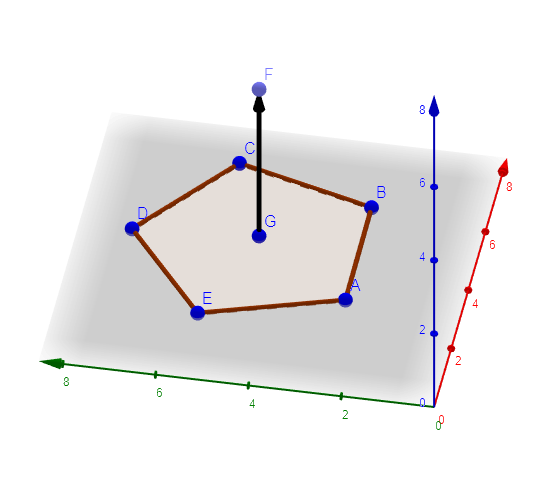
*Фиг 3.37 Разделяне на форма, която не образува затворен контур*

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

* + - 1. **GetPoints**

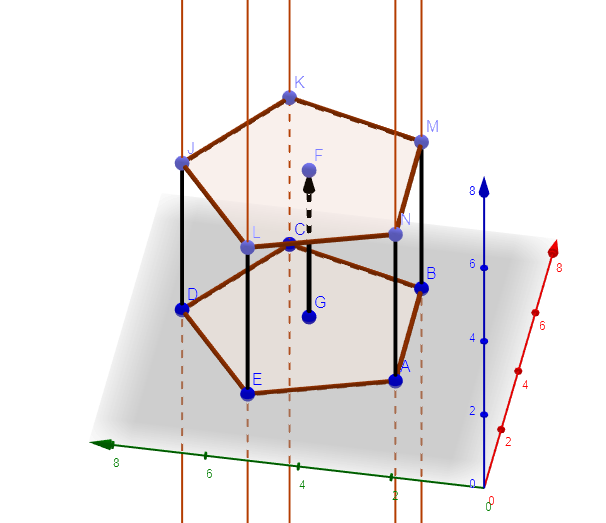
Метода GetPoints приема списък с линии и връща списък с всички точки помежду им. Може да се включи и проверка, която филтрира точките, премахвайки дупликатите. Следователно списъка, който бива връщан, може да съдърща както всички точки след разделянето на формата на линии, така и само индивидуалните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

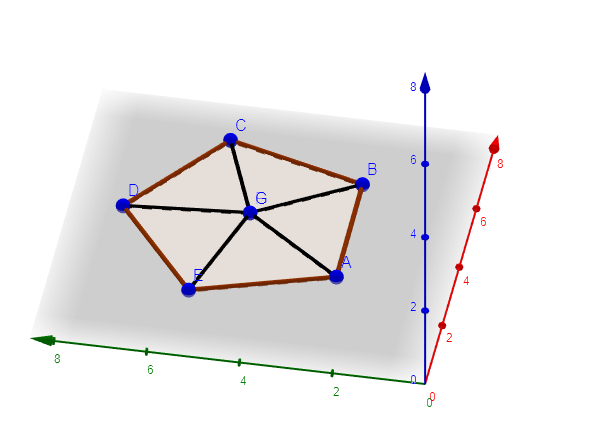
*Фиг 3.38*

* + - 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 което се транслират по посока вектора с неговата дължина (Фиг. 3.39).

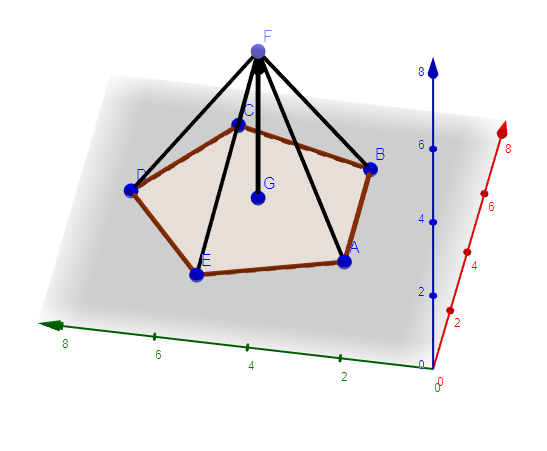
*Фиг 3.39*

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 откриван центъра на определена форма (Фиг. 3.41).

****

*Фиг 3.41*

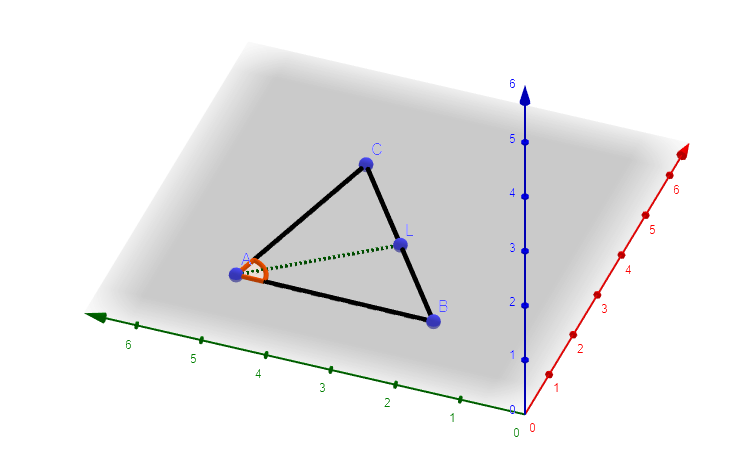
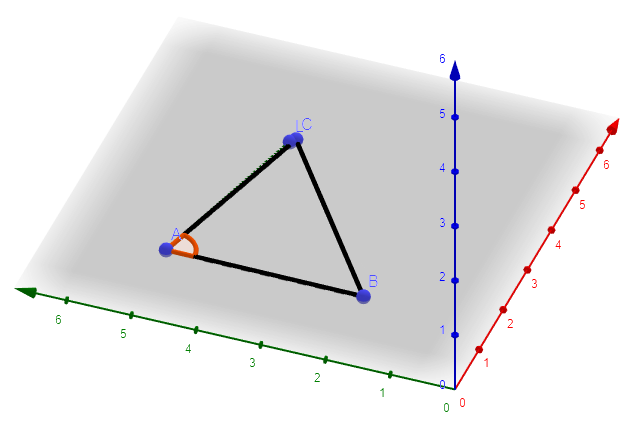
*Фиг 3.40*

****

* + 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:

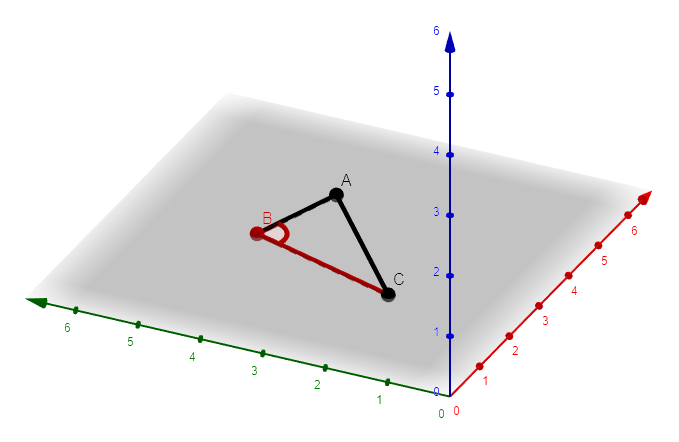
От чертежа може да заключим че , или ако прехвърлим получаваме че . След заместване в горната формула вече имаме че

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**

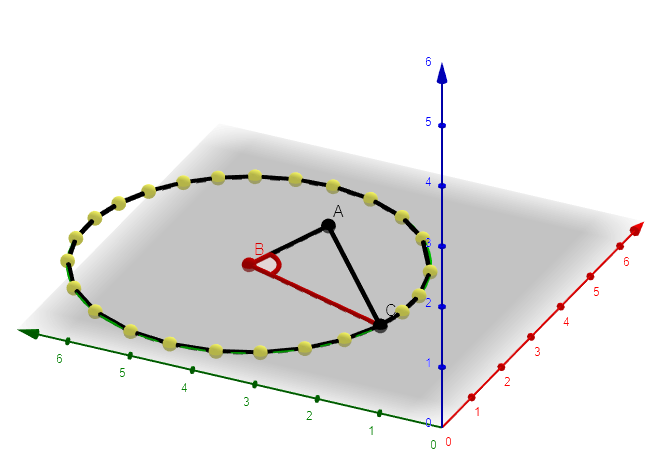
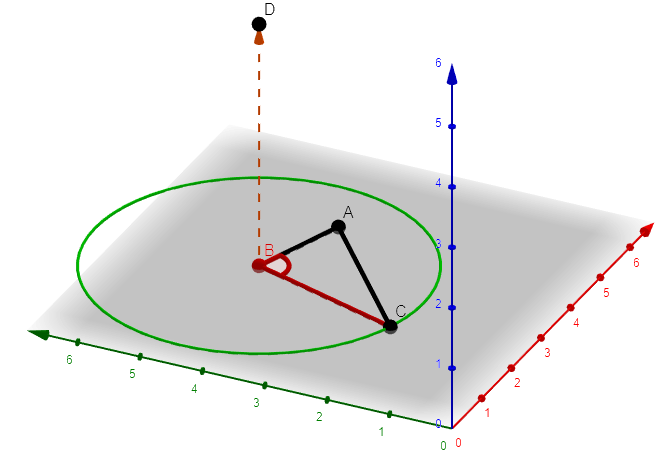
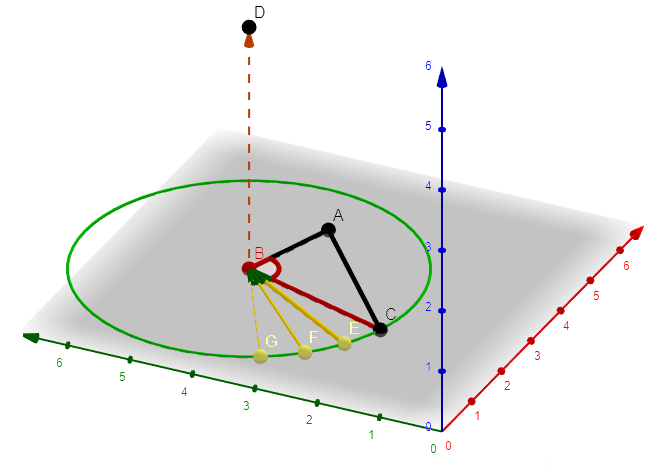
При селектиране на точка, линия и ъгъл една единствена окръжност бива дефинирана. Тази окръжност бива създадена с център селектираната точка, радиус – селектираната права и ротация – избрания ъгъл.

*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**