ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ

към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ДИПЛОМНА РАБОТА

Тема: **Софтуер за 3D визуализация на проектни решения за селищни канализационни мрежи.**

Дипломант:

Кристиян Попов

Научен ръководител:

гл. ас. инж. Росен Петков

СОФИЯ

2015

ТЕХНОЛОГИЧНО УЧИЛИЩЕ
ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Дата на заданието: 10.11.2014 г. Утвърждавам:.....
Дата на предаване: 10.02.2015 г. /проф. д-р инж. Т. Василева/

ЗАДАНИЕ
за дипломна работа

на ученика Кристиян Илиянов Попов от 12 Б клас

1.**Тема:** Софтуер за 3D визуализация на проектни решения за селищни канализиционни мрежи.

2.**Изисквания:**

- Входен интерфейс – включва прочитане на предоставени входни данни и проверка за коректност;
- Графичен интерфейс за потребителя – Потребителят да може да отваря файлове с данни и да ги редактира;
- Изчислителна част – чрез математически изчисления се създава триизмерен модел на канализационната мрежа;
- Визуализация – изобразяване на модела и способност за неговото разглеждане от различни гледни точки;
- Симулация на работата на системата при нейното натоварване за зададен интервал от време;
- Приложението да работи под MS Windows

Дипломант :.....

Консултант:.....
/ инж. Ст. Дарачев /

Ръководител:.....
/ гл.ас. инж. Р. Петков /

Директор:.....
/ гл.ас. д-р инж. Ст. Стефанова /

Увод

Канализационната система е една важна част от инфраструктурата на населените места. Тя е предназначена да отвежда дъждовните и отпадъчните води извън населеното място. Канализацията има важно значение за екологията в района и за хигиената на населението. Добре изградената канализация, спомага за предотвратяване на последствията на наводнения при обилни валежи, снеготопене и др.

Канализационни системи са съществували още от дълбока древност, за което свидетелстват археологическите открития в древни градове. С развитието на човешката цивилизация и концентрирането на населението в градовете, канализацията се превръща в абсолютна необходимост. Съгласно действащата нормативна уредба в България, всички населени места с повече от 5000 жители, трябва да имат изградена селищна канализационна мрежа. В много от малките селища предстои изграждането на такива системи.

Изграждането на канализационна система е отговорна задача, свързана с немалък разход на труд, материали и финансови средства. Канализацията трябва да е съобразена с гъстотата на населението, релефа на терена, климатичните особености на района и други фактори. Канализационната мрежа може да се разглежда като съвкупност от тръби и шахти, свързани по определен начин в клонове, които елементи отговарят на съответните технологични критерии. При проектирането на нови и обследването на

съществуващи селищни канализационни мрежи, строителните инженери използват различни методи и софтуер, в резултат на което получават числови и двумерни графични данни за решението (таблици и чертежи). До момента не са използвани методите на триизмерното визуализиране в проектантската дейност.

Целта на настоящата дипломна работа е разработката на софтуер за 3D визуализация, който да предостави на проектантите триизмерен (обемен) модел, чрез който да придобият както детайлен, така и цялостен поглед върху проекта. Чрез 3D визуализацията на системата, значително се улеснява откриването на грешки, както и подробно наблюдение на сложни участъци. Практическото използване на този софтуер ще доведе до спестяването на средства при проектирането на една такава система.

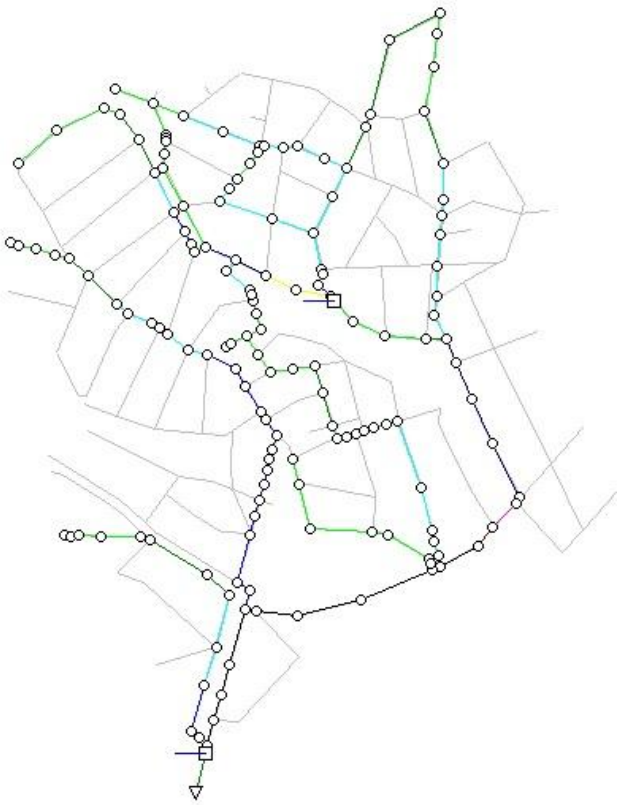
Първа глава

Технологии за графично изобразяване на канализационни системи и развойни средства за 3D визуализация

1.1 Преглед на съществуващите технологии за графично изобразяване на канализационни системи

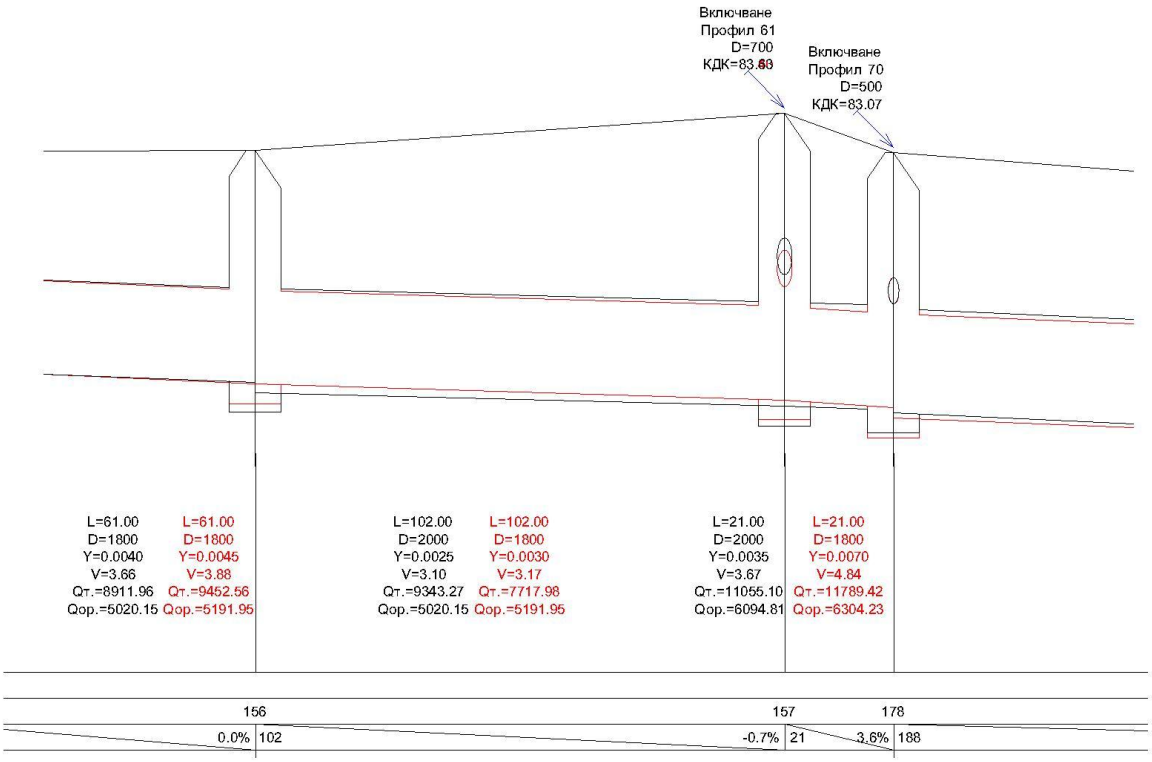
Съществуват три типа софтуер, които в момента се използват при проектирането и обслужването на канализационните мрежи. Това са:

● Географски информационни системи (ГИС) – Дават план на разположението на елементите на канализационната система в рамките на плана на населеното място. Примерен изглед на такъв план е показан на фиг. 1.1



Фиг. 1.1

● Оразмерителни системи – използват се при проектиране на нови канализационни системи. В резултат на получените данни (обикновено в табличен вид) проектантите изчертават елементите на канализационната мрежа (клонове и участъци) с помощта на програми като AutoCAD. На фиг. 1.2 е изобразено изчертаването на участък, а на фиг. 1.3 – на цял клон от канализационна система.



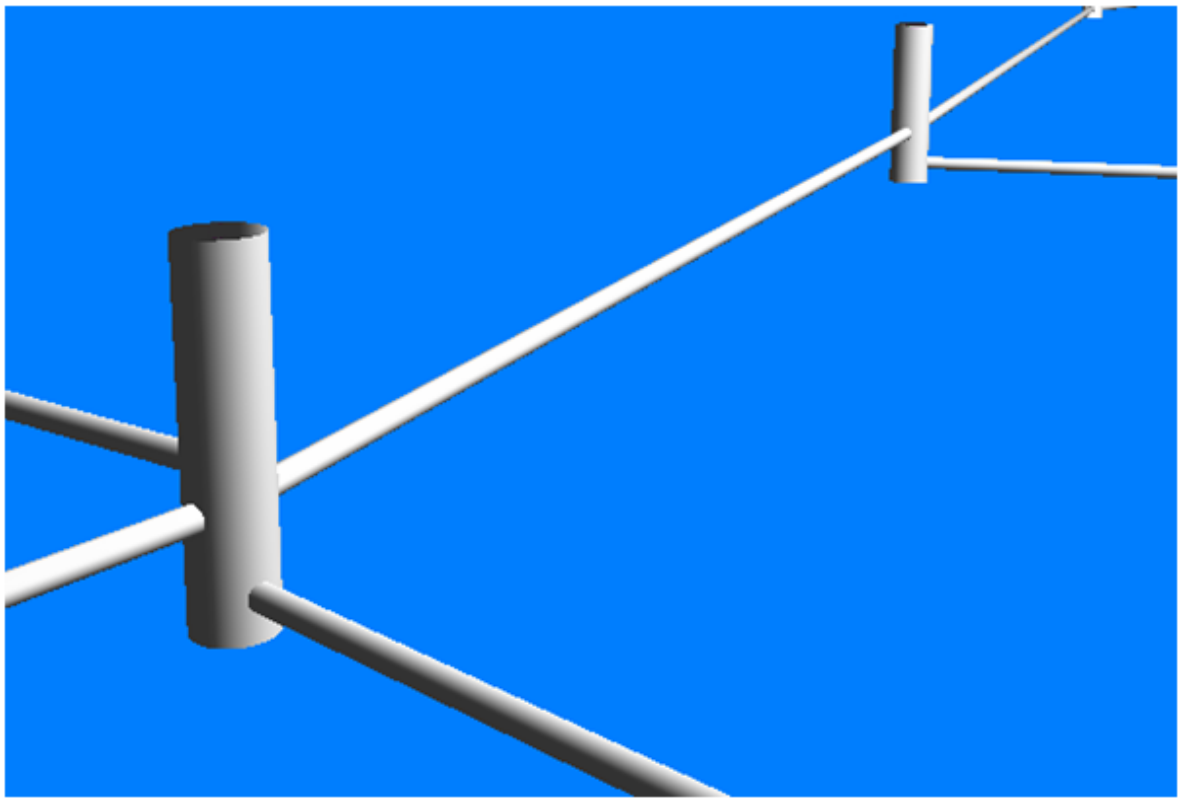
Фиг. 1.2

Фиг. 1.3

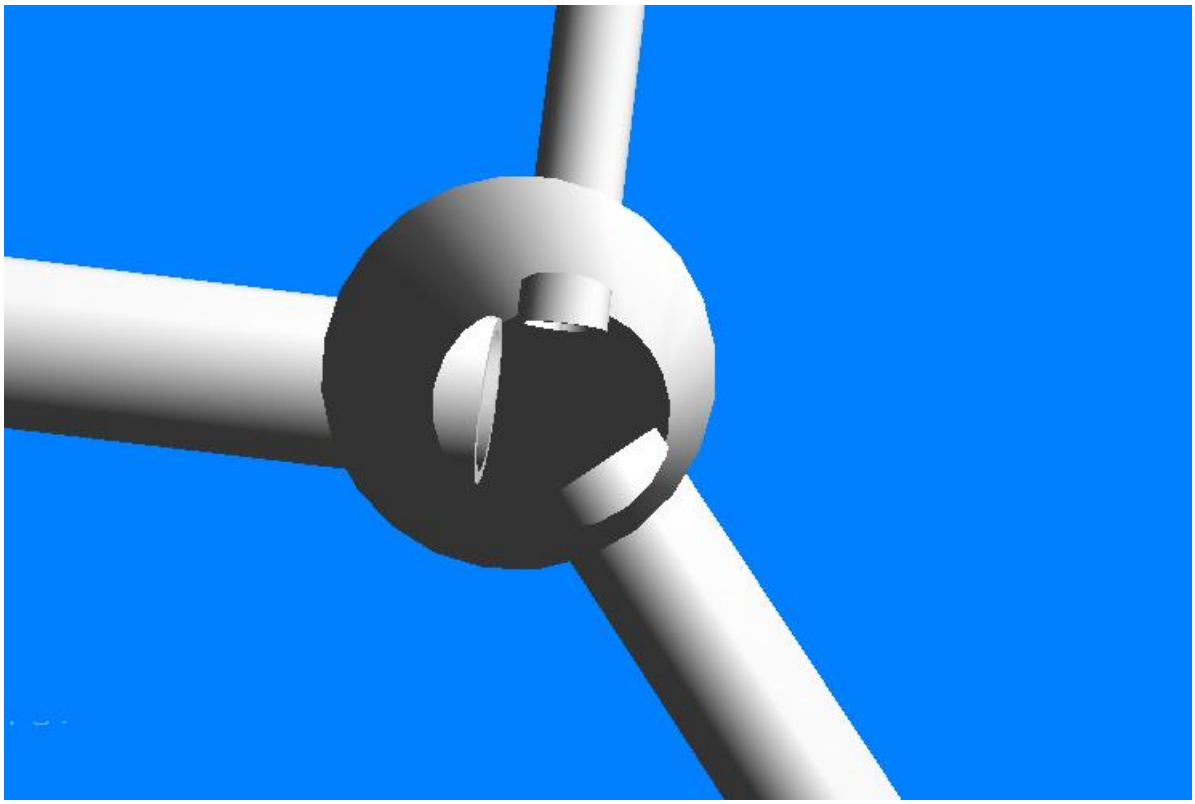
- Симулационни системи – също се използват от проектантите със цел да симулират натоварването на канализационната система в отделни участъци и като цяло с цел проверка на нейния капацитет и възможности. Такива програми са специално изработени от научни институти и организации и са изключително скъпи. Графичното изобразяване на симулираните процеси е в 2D анимация.

Без да е практика 3D визуализация на канализация, може да се създаде с помощта на AutoCAD, чрез ръчно въвеждане на данните за изчертаване. Това е труден процес, който отнема много време.

От направения преглед на съществуващите технологии за графично изобразяване на канализационни системи, става ясно че липсва софтуер за 3D визуализация, който да допълни изгледа върху системата, да подпомогне и ускори проектантския труд. Примерен изглед на 3D модел на елементи от такава мрежа са показани на фиг. 1.4 и 1.5.



Фиг. 1.4



Фиг. 1.5

1.2 Преглед на развойните средства за 3D визуализация

За създаването на 3D визуализация са нужни сериозни познания в областта на 3D графиката и нейните механизми. Има два основни приложно-програмни програмни интерфейса (API) за създаването на 3D графични приложения, а това са OpenGL и Direct3D.

1.2.1 OpenGL (Open Graphic Library) - Създаден през 1992г, OpenGL е един от най-популярните програмни интерфейси за реализиране на 2D и 3D графика. OpenGL позволява съвместимост с други операционни системи, без разлика в качеството на изображението. Този интерфейс не отстъпва на производителност спрямо Direct3D. Той е напълно свободен за използване и има голям набор от литература и информация за него. Има добра структура и логичен синтаксис. Възможно е използването му в различни програмни езици, като C, C++, C#, Java, Ruby, Perl, Pascal, Delphi и др. OpenGL търпи постоянно развитие. Новите обновления се обявяват предварително, за да могат програмистите да пренапишат своят код съобразно новите възможности на това API. Едно от добрите качества на този интерфейс, е че всяка следваща версия е съвместима с по-старите. Това позволява стабилност в приложенията и имплементации на OpenGL да вървят на машини с различни възможности. Това позволява широка достъпност на програмният продукт. OpenGL се използва широко в Linux платформите и Mac OS.

1.2.2 Direct3D – Част от DirectX API, създадено от Microsoft. Direct X е създаден за Windows95, насочен към разработката на 3D графични приложения и игри. Direct3D за разлика от OpenGL се поддържа единствено на платформите на Windows, Xbox 360. Поддръжката на програми с Direct3D върху други операционни системи се получава чрез емулатори, а не директно, което води до забавяне на изчисленията. За разлика от OpenGL, Direct3D не е съвместим с по-стари версии. Това изисква пренаписване на приложението. Direct3D може да бъде използван в много езици, като C++, Visual C++, C#, Visual Basic. Съществуват голямо количество ресурси за Direct3D, но повечето са платени. Развойната среда е по-опростена и по-лесна за работа. За разработка на софтуер върху Direct3D, съществува SDK, което е свободно и може да бъде изтеглено от официалният сайт на Microsoft.

1.2.3 Програмни езици.

След изчистването на идеята за приложение, се налага да бъде направен избор на програмен език, според изискванията му. Съществуват фактори, като време, труд и производителност на приложението. Когато се създава 3D визуализация, има известни ограничения в езиците, които могат да бъдат използвани. Това се дължи на различните API и платформи. Според тях се определя езика за програмиране. Масово се използва C++ тъй като има добра производителност, обектно ориентиран е, относително прост, има поддръжка от OpenGL или Direct3D, а и много други библиотеки. Java също заема своето място, заради мултиплатформеността на

кода. Използват се също Visual C++, C#, C. По-рядко се използват, Ruby, Python, Pascal.

1.2.4 Други развойни средства

Съществуват програмни продукти за 3D дизайн, които позволяват чрез собствен скриптов език да се автоматизира, генерирането на 3D модел по дадени данни и неговото визуализиране. Такива са 3DS MAX Studio, AutoCAD 3D, Solidworks и др. Характерна черта на тези продукти, е че са много мощни, но същевременно изискват висока квалификация за работа с тях. Има известни ограничения по отношение на автоматизацията при работа с голям обем данни.

Втора глава

Функционални изисквания, избор на развойни средства, алгоритъм и структури

2.1 Функционални изисквания

2.1.1 Входен интерфейс

Чрез входния интерфейс, трябва да се извърши прочитането на файл с данни, подаден от инженера-проектант. Всеки ред от този файл, съдържа данни (координати на начална и крайна точка, наклон, диаметър на тръбите и др.) за всеки един участък от канализационната система. При прочитането на данните се извършва проверка за коректност.

2.1.2 Изчислителна част - изграждане на триизмерен модел

Триизмерният модел на системата, следва да се изгради автоматично на базата на получените входни данни. Това става чрез тяхната обработка с помощта на подходящи математически изчисления.

2.1.3 Визуализация на триизмерния модел

Триизмерният модел на системата трябва да може да се разглежда от потребителя, като се управлява гледната му точка. Гледната точка да може да променя местоположението си с цел по-общ или по-детайлен изглед върху мрежата.

2.1.4 Графичен потребителски интерфейс

Потребителят да има възможност да отваря файлове с данни. Да се предостави възможност за навигация на гледната точка чрез подходящи бутони. Да има обратна връзка с информация за потребителя. Потребителят да има възможност да експортира триизмерният модел във външен файл.

2.1.5 Експорт на триизмерният модел във външен файл.

Генерираният от програмата модел да може да се експортира в стандартен файлов формат с цел съвместимост с други програмни системи.

2.1.6 Симулация

Разработвания продукт да има възможност да симулира работата на системата при нейното натоварване.

2.1.7 Операционна система

Разработваното приложение да работи под MS Windows.

2.2 Избор на развойни средства

2.2.1 Избор на API

След направено проучване на няколко API-та за 3D графика, се взе решение да се стъпи върху OpenGL. Това позволява приложението по-лесно да бъде преработено за други операционни системи, което би довело до по-широкото му използване. OpenGL се е доказал през годините, като стабилен интерфейс поддържан и

непрестанно обновяван много години. Основно предимство, е че има голямо количество литература, която е свободна за използване. По този начин се спестяват средства за разработката на едно приложение с OpenGL. Този интерфейс е добре структуриран и позволява гъвкавост при модификация или обновяване на приложението, според новите нужди на потребителят. OpenGL е съвместим с по-старите хардуерни платформи и операционни системи.

2.2.2 Избор на език за програмиране

Изборът на език е редно да се направи след като е избран вече графичният програмен интерфейс. Един от най-популярните езици за програмиране с OpenGL е C++. В голяма част от свободната литература, този език е използван относно описанието за работа с OpenGL. C++ позволява да се използват и много библиотеки написани за него. Има голямо количество литература за C++, която също е в повечето случаи свободна. Изчистеният и добре структуриран синтаксис позволява изпълнението на алгоритми с по-малко код, спрямо други програмни езици. Има IDE среди, които също са свободни за използване. Придобитите знания и опит върху този език с годините, улесняват работата и спестяват време за изучаване на нов програмен език.

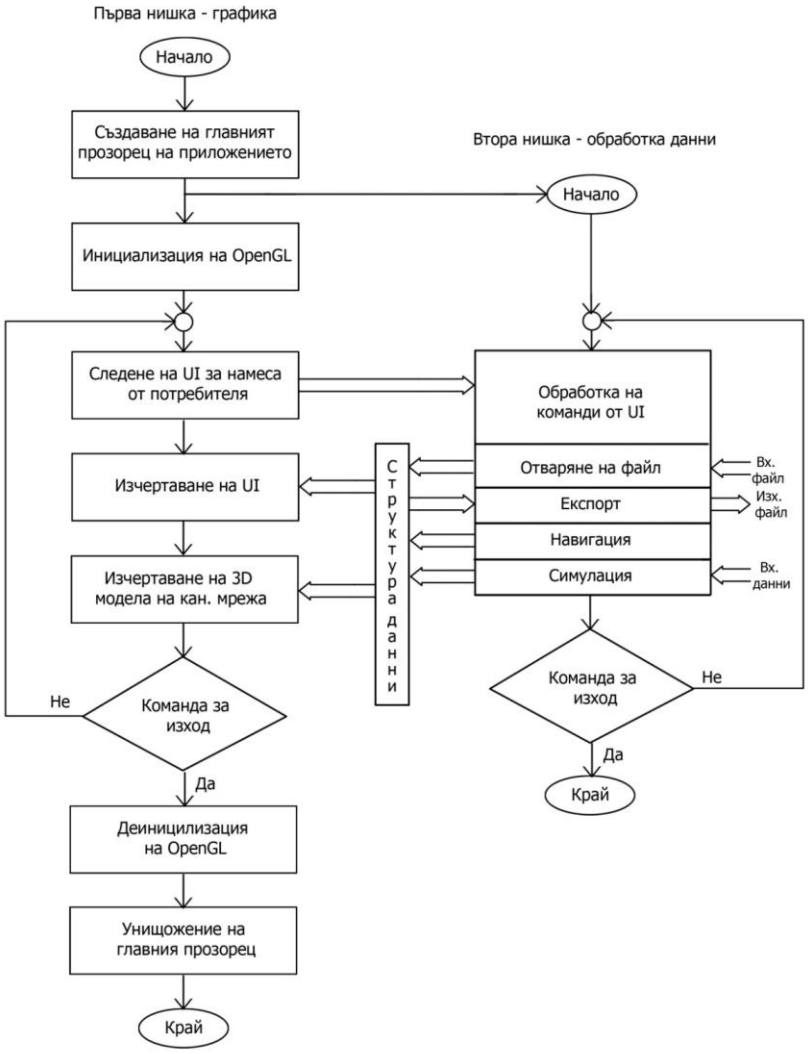
2.2.3 Избор на IDE

Dev C++ напълно задоволява нуждите за разработка на едно приложение с такава цел. Това IDE е напълно безплатно за използване, с достатъчно опростен интерфейс и множество библиотеки. Въпреки че поддръжката му е спряна през 2005 година,

това не пречи на съвместимостта му със други IDE-та, като Microsoft Visual Studio или Code Blocks. Съществуват известни недостатъци и грешки на средата, но те не са от голямо значение за разработваното приложение.

2.3 Алгоритъм и структури

2.3.1 Алгоритъм - На фиг. 2.1 е показана блок схема на алгоритъма на приложението.



Фиг. 2.1

Задачите, поставени пред разработвания софтуер, могат логически да се разделят на два паралелни процеса. Единият е свързан с обработка на данни, подавани от потребителя. Другият процес е графичен, свързан с прехвърлянето на 3D модела в 2D изображение върху екрана, което изображение зависи от избраната гледна точка. Горните два паралелни процеса предопределят избора на програмен алгоритъм с две програмни нишки. Това дава съществени предимства при работата на приложението на компютър с многоядрен процесор.

Основните елементи на алгоритъма са:

Създаване на главният прозорец на приложението – за да функционира програмата е нужно да бъде създаден прозорец, в който да се съдържа потребителския интерфейс и резултата от визуализацията на модела. Той се създава първоначално с определени размери, но се предоставя възможност да бъде манипулиран.

След това е нужно да бъде стартирана втората нишка за обработка на данни.

Първата програмна нишка продължава с:

Инициализация на OpenGL – За да е възможно изчертаването на графика в полето на прозореца, е нужно първоначалното установяване на графичния интерфейс. В тази инициализация се установяват настройките за графиката на приложението, в зависимост драйвера на видеокартата, разделителна способност, цветове и др.

За всяко графично приложение е характерен цикъл, който работи постоянно, докато не бъде дадена команда за изход. В този цикъл се поместват следните елементи:

Следене на потребителския интерфейс за намеса на потребителя – при натиснат бутон от интерфейса или натиснат клавиш от клавиатурата, или движение на мишка, се дава сигнал за това чрез булеви променливи (флагове), които се следят от друго място във втората нишка.

Изчертаване на потребителски интерфейс – динамичното променяне на данните, изисква динамично изчертаване върху екрана. Тогава потребителят получава реална информация за случващото се с данните от визуализацията. Това включва изчертаване на бутони, показване на динамичен текст, ориентация в пространството и др.

Изчертаване на 3D модела на канализационната мрежа – за да може потребителят да разглежда модела от различни гледни точки, е нужно всеки път да се изчертава неговата нова проекция върху екрана. Всяка намеса от потребителският интерфейс, влияе върху 2D изображението върху монитора. Изчертаването на модела става, като се прочете структурата от данни.

Накратко по този начин работи този цикъл в алгоритъма на приложението. Когато командата за изход е подадена, тогава цикълът се прекъсва и настъпват следните операции:

Деинициализация на OpenGL – нужно е преди да се затвори приложението, да бъде освободено място от оперативната памет на

компютъра, както и паметта на видеокартата. Освен това се указва, че програмата няма да използва този графичен интерфейс и че няма какво да се изчертава на екрана.

Унищожение на програмния прозорец – това е нужно да се укаже към операционната система, да унищожи прозорецът на приложението и да освободи заделената памет за изчертаването му.

Край – настъпва физическата смърт на процеса, като се знае че той е завършил успешно.

Паралелно с работата на първата програмна нишка, описана по-горе, работи втората нишка. След нейното стартиране (веднага след създаването на основният прозорец на приложението) работата й протича по следният начин:

Както в първата нишка, тук отново има цикъл, който продължава, докато не бъде подадена командата за изход. Този цикъл съдържа следните елементи:

Обработка на командите от потребителския интерфейс – има няколко основни команди, които се следят от този елемент. Те са:

Отваряне на файл - показва се диалогов прозорец , чрез който потребителят може да избере нужният файл за изграждането на модела. След избора на файл, става изграждането на модела чрез алгоритъм от математически изчисления. Резултатите се записват в глобална структура от данни.

Експорт на файл – показва се диалогов прозорец, чрез който потребителят запазва файл във формат, годен за ползване в други

среди за 3D визуализация. Прочита се структурата от данни и се привежда в необходимия формат.

Навигация – Потребителят може да се движи напълно свободно в пространството. В зависимост указаната посока на движение се отразяват промени в позицията на гледната точка.

Симулация – Чрез подадени външни данни и параметри за симулацията, се оказват промени върху структурата от данни с цел визуализирането на процеса.

По този начин протича работата на този цикъл. След подаване на командата на изход, този цикъл спира да работи и непосредствено след това следва край на програмната нишка.

За да се избегне т.н. „критична секция” (когато две нишки записват данни в една и съща променлива се стига до грешен резултат), нишката за графичното изобразяване само прочита структурата от данни, докато нишката за обработка на данни може да променя съдържанието на променливите.

2.3.2 Структура от данни

Данните с които работи програмата са структурирани в няколко групи:

- Входни данни
- Данни тръби
- Данни шахти
- Данни за местоположението на гледната точка

Входните данни по които се изгражда 3D модела на системата са структурирани в записи съдържащи следните полета:

Име на поле	Кратко описание
№ у-к	Съдържа поредния номер на участъка
Име на клон	Универсален индентификатор на клона
Начална точка	Име на началната точка (име на начална шахта за участъка)
Крайна точка	Име на крайна точка (име на крайна шахта за участъка)
L (хоризонтална дължина)	Дължината на хоризонталната проекция на тръбата
D (условен диаметър)	Диаметър на тръбата по проект [mm]
Y (наклон на тръбата)	Наклон на тръбата [%]
Начална точка-КТ	Надморска височина на терена на началната точка [m]
Крайна точка-КТ	Надморска височина на терена на крайната точка [m]
Кота дъно канал-НТ	Надморска височина на мястото за връзка с тръбата с шахтата в началната точка [m]
Кота дъно канал-КТ	Надморска височина на мястото за връзка с тръбата с шахтата в крайната точка [m]
Начална точка-Х	X координата на началната точка [m]
Начална точка-У	Y координата на началната точка [m]
Крайна точка-Х	X координата на крайната точка [m]
Крайна точка-У	Y координата на крайната точка [m]
Вътрешен диаметър	Вътрешен диаметър на тръбата [mm]
Външен диаметър	Външен диаметър на тръбата [mm]
Дебелина на стената	Дебелина на стената на тръбата [mm]

Както е написано по-горе, канализационната система е изградена от тръби и шахти, които отговарят на определени технологични изисквания. След обработката на входните данни, резултатът се записва в масиви от данни за всяка тръба и шахта. Тръбите, като обект имат следните параметри в които се записват данни:

Параметър	Описание
Номер на участък	Универсален идентификатор за участъка
Вътрешен диаметър	Вътрешен диаметър на тръбата в [mm]
Външен диаметър	Външен диаметър на тръбата в [mm]
Дебелина на стената	Дебелина на стената на тръбата в [mm]
Дължина	Абсолютна дължина на тръбата в [m]
Хоризонтална дължина	Дължината на хоризонталната проекция на тръбата [m]
Наклон	Наклон на тръбата [%]
Ъгъл	Абсолютен ъгъл в пространството на тръбата по проекция изглед отгоре.
Начална точка X	Началната точка на тръбата по X координати [m]
Начална точка Y	Началната точка на тръбата по Y координати [m]
Начална точка Z	Началната точка на тръбата по Z координати [m]
Крайна точка X	Крайна точка на тръбата по X координати [m]
Крайна точка Y	Крайна точка на тръбата по Y координати [m]
Крайна точка Z	Крайна точка на тръбата по Z координати [m]

Шахтите, като обект имат следните параметри:

Име на поле	Кратко описание
Име на шахта	Универсален идентификатор на шахтата
Местоположение X	Съдържа местоположението на шахтата по X [m]
Местоположение Y	Съдържа местоположението на шахтата по Y [m]
Местоположение Z терен	Съдържа надморската височина на която се намира капака на шахтата [m]
Местоположение Z дъно	Съдържа надморската височина на която се намира дъното на шахтата [m]
Диаметър	Диаметър на шахтата

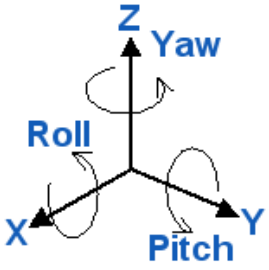
Диаметърът на шахтата се определя в зависимост вътрешния диаметър на най-широката тръбата за текущата шахта. Това става според следната таблица:

Диаметър тръба	Диаметър шахта
До 600mm	1000mm
До 1000mm	1500mm
До 1500mm	2000mm
Над 1500mm	3000mm

Изграждането на един триизмерен модел става, чрез построяването на прости геометрични фигури (триъгълници и четириъгълници), които се свързват по между си и образуват мрежа. Всеки обект от типа шахта или тръба след обработка на входните данни, съдържа в себе си информация за изграждането на модела. Съдържат се масиви от точки с определено местоположение, данни за налагане на изображение върху модела (текстура), данни за осветеност на повърхнините на модела (нормали) и алгоритъм за свързването на точките в модела.

Нужно е също да има данни за местоположението на гледната точка, от която се разглежда модела, за да се покаже правилно на екрана.

Те се свеждат до местоположение по осите X,Y и Z в триизмерното пространство и ъгли на завъртане по съответните оси. Чрез движението на курсора на мишката върху екрана, от 2D координатите му, ъглите се изчисляват в полярни (yaw, pitch), а след това в ъгли на осево завъртане. (Фиг. 2.2)



Фиг. 2.2

Трета глава

Реализация на приложението

3.1 Основни класове и структури

Проектът е реализиран на няколко основни файла:

Main.cpp – съдържа основния алгоритъм на приложението и функции за изчертаването на модела.

Engine.h – съдържа класове свързани с управлението и модела на системата

SEW-3D.h – съдържа класове и функции свързани с изграждането на модела.

Menu.h – съдържа класове свързани с управлението на потребителския интерфейс.

dds.h – служи за зареждане на изображения от външни файлове.

Освен това се използват допълнителни библиотечни файлове, които се свързват към основните. Те са свободни и позволяват използването на повече функции на OpenGL.

Engine.h е написан в резултат на работата върху друг авторски продукт (SAD engine I), който е в процес на разработка. SAD (Simulations Adapted for Display) engine има за цел да бъде платформа за визуализация на различни природни процеси, симулации на различни системи, машини. Чрез бъдещето развитие на тази платформа, това би довело до улеснение при създаването

на подобен софтуер за визуализация, обект на разглежданата дипломна работа.

В приложението са заложени част от основните класове използвани при SAD Engine.

Движението на обект в пространството е резултат на преместване на определено разстояние за определено време. Времето е относително за различните обекти в пространството, но без него няма как да съществува света около нас. Това приложение има за цел да даде информация свързана с нещо от реалният свят, следователно е дефиниран класа „timer”.

Класа timer е изключително просто устроен. Използва се, когато искаме да отброяваме някакво изминало време. Той има следните методи:

void reset() – занулява стойността на таймера

void set(int new_time) – задава стойност на таймера

void inc() – увеличава стойността на таймера с едно

int get() – връща стойността на таймера

Мерната единица е условна в класа и се определя от главната програма. Ако се отмерват минути, то на всяка изминала минута се подава сигнал към таймера да се увеличи с едно. Прочитането на стойността на таймера в определен момент дава изминалото време в зададената мерна единица.

По нататък е дефиниран класа „main_object”, който дефинира обект в пространството. В него се съдържат координати на обект,

размери и някои основни природни закони, като свободно падане и др.

```
class main_object{  //Главен обект
    float x,y,z;  //Координати на обекта
    float angle_x,angle_y,angle_z;  //Ъгли под които е завъртян обекта спрямо осите на координатната система
    float mass;  //Маса на обекта
    float lenght, width, height;  //Габаритни размери
public:
    ..... //Методи свързани с достъп до данните на обекта (getters and setters)
    void phys_fall(float Vo,float G,int timer){  //Метод в който е заложена формулата за свободно падане
        z+=Vo/1000 - (G*timer)/2;
    }
};
```

За да съществува визуализацията е нужно определянето на гледната точка (камера). Това е решено с класа „camera”, който наследява класа “main_object. Класа „camera” има следните методи:

```
float get_elevation()  //връща ъгъл спрямо хоризонта
void set_angles(float azimuth,float elevation)  //задаване на ъглите на камерата, като първият параметър е ъглово отклонение спрямо север, а втория задава ъгъл спрямо хоризонта
void set_coords(float new_x,float new_y, float new_z) //задава координатите на камерата
```

Това определя логически гледната точка, но за да задоволи изискванията на продукта, да съществува възможност за навигация

е написан друг клас „spectator”, който наследява класа „camera”. В него се съдържат методите за навигация:

```
void move_forward(float speed){
        set_y(get_y()+speed*cos(get_angle_z()*PI/180));
        set_x(get_x()+speed*sin(get_angle_z()*PI/180));
}  //дава се команда за навигация напред по вектора на камерата, като се задава параметър скоростта на камерата. Отново скоростта е условна и зависи от мерната единица за времето
```

Следващите три метода работят по аналогичен начин по вектора на камерата и отново се задава параметър скорост.

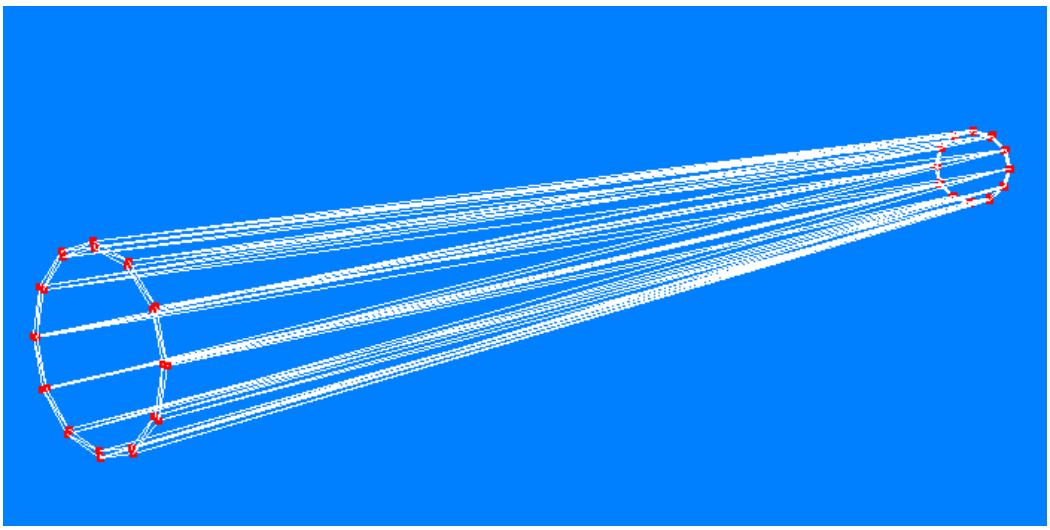
```
void move_back(float speed) //движение назад
void move_left(float speed) //движение наляво
void move_right(float speed) //движение надясно
```

Следващите методи се състоят само от движение по Z оста.

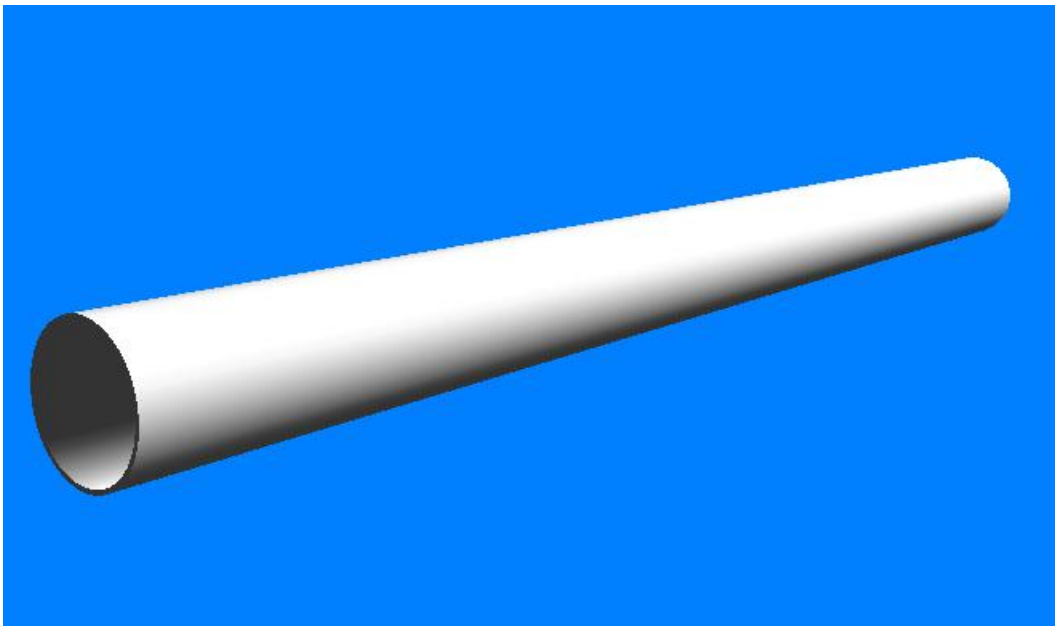
```
void move_up(float speed){
        set_z(get_z()+speed);
} //движение нагоре по Z
void move_down(float speed) //движение надолу по Z
```

Една визуализация няма как да съществува без триизмерен модел. Моделът е съставен от множество прости геометрични фигури (полигони) свързани в мрежа, която обхваща всичките му повърхнини. В зависимост от осветеността и изображенията наложени върху тях, може да се създаде илюзия за реалистичност на повърхнината. В конкретното приложение е избрана работата с най-простата геометрична фигура определяща равнина – триъгълник. Една триъгълна мрежа на един модел би изглеждала,

така както е показано на Фиг. 3.1, а в резултат при изобразяване да се получи изображението показано на Фиг. 3.2



Фиг. 3.1



Фиг. 3.2

Мрежата на модела се изгражда чрез масив, който указва в какъв ред точките се свързват. Например за да се дефинира триъгълник се нуждаем от 3 точки, 3 текстурни координати, 3 нормали. Това означава че всеки връх има своя точка, своя текстурна координата и своя нормала. Това е решено чрез тази структура:

```
struct f_point{
    int v, t, n;
};
```

Където v е ID на точката, t – ID на текстурна координата и n – ID на нормала. Според всяко ID се търси в масива от данни, кой елемент съответства на него.

Следващата структура показва как се описва повърхнината (полигона)

```
struct face{
    f_point a,b,c;
};
```

Съответно a, b, c означават върховете на триъгълника. Например с a.v ще се обърне към ID на координатите на точка a от триъгълника.

Класа „model” е един от най-важните в приложението, обектите тръби и шахти го наследяват. Той съдържа променливи и масиви с елементите на модела, които са публични. Т.е. могат да бъдат достъпвани свободно.

```
    face* f; // Масив от полигони
    float* arr_v; //Масив от координати на точките
    float* arr_vt; //Масив с текстурни координати
    float* arr_vn; //Масив с нормали
    int num_f; //Брой полигони
    int num_v; //Брой точки
    int num_vt; //Брой текстурни координати
    int num_vn; //Брой нормали
```

Файлът SEW-3D.h съдържа основните структури и алгоритми за изграждане на модела на системата.
Данните, получени от входния файл по които се създава модела, след обработка се съхраняват в масив от следната структура:

```
struct data{
    int num; //пореден номер на участък
    char* name_branch; //име на клон
    char* name_start_p; //име на начална точка
    char* name_end_p; //име на крайна точка
    float hor_length; //дължина на хоризонталната проекция на тръбата
    char* cond_d; //условен диаметър
    float slope; //наклон
    float start_z_terrain; //надморска височина на терена на начална точка
    float end_z_terrain; //надморска височина на терена на крайна точка
    float start_z; //надморска височина на мястото за връзка с тръбата с шахтата
                   в началната точка
    float end_z; //надморска височина на мястото за връзка с тръбата с шахтата в
                   крайната точка
    float start_x; //местоположение на началната точка по X (локално)
    float start_y; //местоположение на началната точка по Y (локално)
    float end_x; //местоположение на крайната точка по X (локално)
    float end_y; //местоположение на крайната точка по Y (локално)
    float tube_inner_d; //вътрешен диаметър на тръбата
    float tube_outer_d; //външен диаметър на тръбата
    float tube_thickness; //дебелина на тръбата
};
```

Във входните данни е използвана геодезична координатна система. Геодезичните координати са в метри с точност до третия знак след десетичната точка, като X е източна дължина, Y е северна ширина и Z е надморска височина. Тъй като координатите по X и Y могат да

бъдат прекалено големи числа, за да се поемат от стандартните типове (float, double), е изградена структура от две променливи, съответно за цялата и дробната част.

```
struct coords_fix{
   long int a; //съдържа записа на дробното число преди десетичната точка
   double b;  //съдържа записа на дробното число след десетичната точка
};
```

Създадена е специална функция за обработка на координатите:

```
SepFloat(char* field, coords_fix &cf);
```

Подават се два параметъра. Единият е поле от запис на входния файл, съдържащо в символен низ съответната координата. Резултатът във формата на по-горе написаната структура се записва в другия параметър „cf”

Функцията чете полето до десетичната точка, и записва първата част в променливата „cf.a”, като се прехвърля от символен низ (char*) в тип „long int”, а остатъка след десетичната точка се записва във втората променлива „cf.b” в тип „double”.

Създадени са две функции, които са основни за изграждането на триизмерен модел на ротационни тела.

```
int BuildCircle(float diameter,float step,float* &points){
    points = (float*)malloc(sizeof(float));

    int counter=0;

    for(float i=0; i<360;i+=step){
        points=(float*)realloc(points,(counter+2)*sizeof(float));
        points[counter]=cos(i*PI/180)*diameter*0.5*0.001;
        points[counter+1]=sin(i*PI/180)*diameter*0.5*0.001;
        counter+=2;
    }
    return counter;
}
```

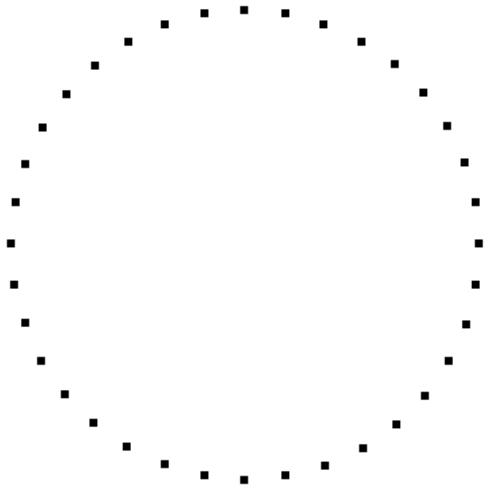
Функцията „BuildCircle” създава точки върху окръжност, получени в следствие на ротация през определена стъпка в градуси. Местоположението им се смята по следните формули:

$$\mathbf{x} = cos(\alpha).\frac{d}{2}$$

$$\mathbf{y} = sin(\alpha).\frac{d}{2}$$

d – диаметър на окръжността; α – ъгъл на ротация.

На фиг. 3.3 е показан графичният резултат от работата на функцията „Build Circle”:

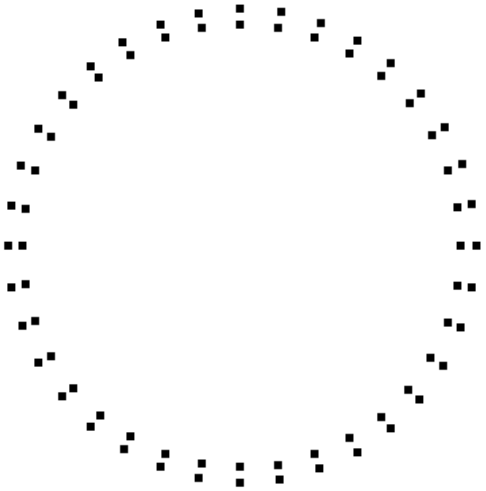


Фиг. 3.3

BuildRing(float inner_d,float outer_d,float step,float* &points)

Функцията „BuildRing” създава точки върху пръстен (две концентрични окръжности) по аналогичен начин на „BuildCircle”. Като параметри се подават вътрешния и външния диаметър на пръстена, стъпката на ротация в градуси и масив от координатите

на получените точки. Функцията „BuldRing” използва два пъти в себе си функцията „BuildCircle”.
На фиг. 3.4 е показан графичният резултат от работата на функцията „BuildRing”:



Фиг. 3.4

В файла SEW-3D.h се намират двата основни класа за изграждането на системата „pipe” и „shaft” (тръба и шахта).

Класът „pipe” съдържа в себе си данните за изграждането на триизмерен модел на тръба и неговото позициониране в пространството. Още съдържа в себе си няколко метода за достъп до данните и един основен метод за построяване на модела. Класът „pipe” наследява класа „model” от файла „engine.h”.

```
class pipe: public model{
   char* clone_name; //име на клон
   float inner_d,outer_d; //вътрешен и външен диаметър
   float thickness; //дебелина на стената на тръбата
   float abs_length; //абсолютна дължина на тръбата в пространството
   hor_length; //дължина на тръбата по хоризонтална проекция
   float slope; //наклон на тръбата в проценти
   float top_angle; //ъгъл на тръбата в проекция изглед отгоре
```

```
    float start_x,start_y,start_z; //координати на начална точка
    float end_x,end_y,end_z; //координати на крайна точка
    float shaft_d; //диаметър на шахтата за връзка

public:

….  //Съдържат се методи за достъп до данните

    void build_pipe(  //задават се параметри за построяване на модела на тръбата
        char* new_clone_name,
        float new_inner_d,
        float new_outer_d,
        float new_hor_length,
        float new_slope,
        float new_start_x,
        float new_start_y,
        float new_start_z,
        float new_end_x,
        float new_end_y,
        float new_end_z,
        float new_shaft_d)
    {
        …..

    }

        …..

};
```

В този метод се изчисляват:

Абсолютна дължина на тръбата (abs_length)

$$abs\_length = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Хоризонтална дължина на тръбата (проекция на тръбата от изглед от горе (hor_length)

$$hor\_length = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Абсолютен ъгъл на завъртане (top_angle) на тръбата по Z (X+ се приема за ъгъл 0)

$$\text{top\_angle} = arccos\left(\frac{x_2 - x_1}{hor\_length}\right)$$

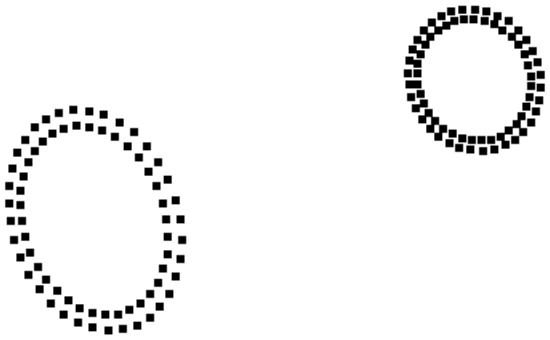
Ъгъл на ротация по Z (rot_angle)

$$rot\_angle = \begin{cases} top\_angle, & \text{при } y_2 \geq y_1 \\ -top\_angle, & \text{при } y_2 < y_1 \end{cases}$$

Наклон на тръбата (slope)

$$\text{slope} = arcsin\left(\frac{z_2 - z_1}{abs\_length}\right)$$

След определяне на горните стойности, метода извиква два пъти функцията „BuildRing" за създаване на точките от модела в двата края на тръбата (Фиг.3.5).



Фиг. 3.5

Изграждането на триизмерния модел на тръбата в пространството, започва като тръбата се построява хоризонтално и ориентирана по оста Y, където началната точка е с локални координати [X=0, Y=0, Z=0]. От двата края тръбата се скъсява, така че подава в шахтата на 20 см от нейната стена (по изискване на методите за проектиране). Новите координати за краищата на

модела на тръбата се изчисляват за всяка точка на модела, както следва:

$$y_1' = y_1 + \frac{D}{2} - 0.2$$

$$y_1' = y_1 + \frac{D}{2} - 0.2$$

Тъй като при построяване $y_1 = 0$, горното може да бъде пресметнато по следния начин:

$$y_1' = y_1 + \frac{D}{2} - 0.2$$

$$y_2' = abs\_length - \frac{D}{2} + 0.2$$

Следва ротация на тръбата по оста X с цел получаване на нейния наклон.

При това действие Y и Z координатите на всяка точка от модела ще се променят. За да се изчислят новите стойности на Y и Z координатите се прави следното:

Изчислява се разстоянието от всяка точка на модела до началното на локалната координатна система по проекция YZ равнината.

$$distance = \sqrt{y^2 + z^2}$$

Новите координати се изчисляват по следният начин:

$$y = cos\left(\arcsin\left(\frac{z}{distance}\right) - \text{slope}\right).\text{distance}$$

$$z = sin\left(\arcsin\left(\frac{z}{distance}\right) - \text{slope}\right).\text{distance} + z_1$$

В последната формула е добавена z координатата на началната точка (z_1), с което точките на модела се изместват

вертикално до реалното местоположение на модела на тръбата по височина (транслация по Z).

Следва ротация по оста Z и транслация по X и Y с цел правилното разполагане на модела на тръбата в пространството.

По аналогичен начин се изчислява разстоянието от всяка точка на модела до началното на локалната координатна система по проекция XY равнината.

$$distance = \sqrt{x^2 + y^2}$$

Новите координати се изчисляват по следният начин:

$$x = sin\left(\arccos\left(\frac{x}{distance}\right) - \text{rot\_angle}\right).\text{distance} + \text{x}_1$$

$$y = cos\left(\arccos\left(\frac{x}{distance}\right) - \text{rot\_angle}\right).\text{distance} + \text{y}_1$$

За да бъде завършен триизмерният модел, трябва да бъдат зададени векторите на нормалите за всяка точка. Нормалите служат правилното осветяване на модела, съобразено с източника на светлина и гледната точка. Осветяване на модела се изчислява чрез функции на OpenGL, като осевите проекции на векторите на нормалите се подават към тази функция, като параметри:

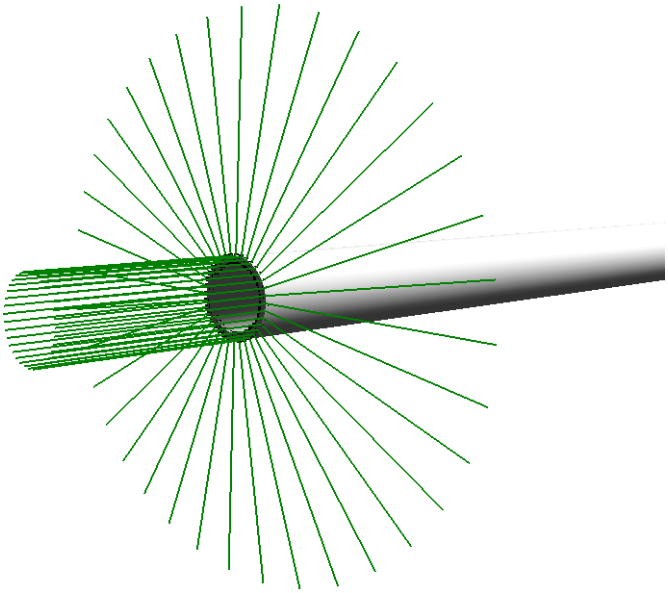
glNormal3d(normal_x, normal_z , normal_y);

Векторът на нормалата и неговите проекции се изчисляват предварително са елемент от модела. Самото изчисляване на нормалите става по следния начин:

$$normal\_x = \cos(\alpha).\sin(-rot\_angle)$$

$$normal\_y = \begin{cases} \sin(\alpha).\sin(slope) + \cos(\alpha).\cos(rot\_angle), & \text{при } y_2 \geq y_1 \\ -\sin(\alpha).\sin(slope) + \cos(\alpha).\cos(rot\_angle), & \text{при } y_2 < y_1 \end{cases}$$

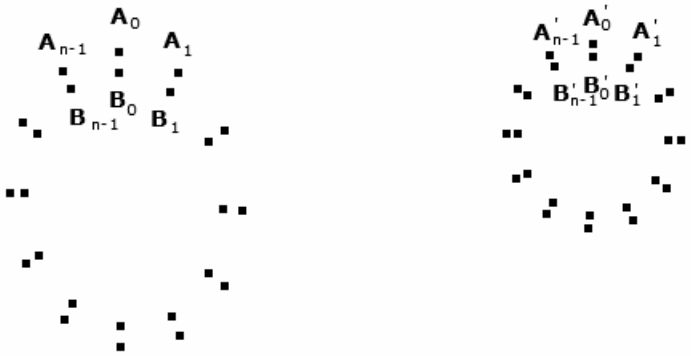
$$normal\_z = \sin(\alpha).\cos(slope)$$



Фиг. 3.6

На фиг. 3.6 са показани графично векторите на така построените нормали

Следва построяването на мрежата от полигони (polygonal mesh). За целта се описва поредица от полигони, дефинирани по серия от точки. В случая полигоните са съставени от 3 точки (триъгълници). Ако означим условно точките от модела на тръбата, както е показано на фиг. 3.7, алгоритъма за свързването на точките може да се опише с таблица 3.1



Фиг. 3.7

i = 0 .. n - 2	i = n - 1
$A_i\ A_{i+1}\ B_i$	$A_{n-1}\ A_o\ B_{n-1}$
$B_i\ B_{i+1}\ A_{i+1}$	$B_{n-1}\ B_o\ A_o$
$A'_i\ A'_{i+1}\ B'_i$	$A'_{n-1}\ A'_o\ B'_{n-1}$
$B'_i\ B'_{i+1}\ A'_{i+1}$	$B'_{n-1}\ B'_0\ A'_0$
$A_i\ A_{i+1}\ A'_{i+1}$	$A_{n-1}\ A_0\ A'_0$
$A'_i\ A'_{i+1}\ A_i$	$A'_{n-1}\ A'_0\ A_{n-1}$
$B_i\ B_{i+1}\ B'_{i+1}$	$B_{n-1}\ B_0\ B'_0$
$B'_i\ B'_{i+1}\ B_i$	$B'_{n-1}\ B'_0\ B_{n-1}$

Табл. 3.1

Така моделът на една тръба е изграден (фиг.3.2) с реалните му размери и местоположение в пространството. Това се повтаря в цикъл за всички тръби от записите на входният файл.

Класът „shaft" съдържа в себе си данните за местоположението на шахтата в пространството, както нейния диаметър и надморската височина на дъното.

```
class shaft: public model{
   float x,y; //местоположение по X и Y
   float z_terrain,z_bottom; //кота терен , кота дъно
   char* name; //име на шахтата
   float diameter; //диаметър
public:
....
//Методи за достъп до данните (getters and setters)

   bool operator==(const shaft& sh){  //Оператор за сравнение между два обекта
      return ((this->x==sh.x)&&(this->y==sh.y)&&(this->z_terrain==sh.z_terrain));
   }

   void set_params(char* new_name, float new_x, float new_y, float new_z_terrain, float
new_diameter){ //Задаване на всички параметри на шахтата
      name=new_name;
      x=new_x;
      y=new_y;
      z_terrain=new_z_terrain;
      diameter=new_diameter;
   }
};
```

В класа е дефиниран оператор за сравнение между два обекта от типа „шахта”, които се сравняват по параметри x, y и кота терен (z_terrain). Този оператор се използва за да се избегне излишното дублиране на шахти в системата, тъй като една шахта може да е обща за няколко участъка.

Функцията set_params() служи за директното задаване на параметрите на класа „shaft”.

Във файла SEW-3D.h е включена функцията

ReadingData (char* file_name,data* &lines,int &progress, float &offset_z),

която обработва данните от входния файл, който се задава като параметър.

Обработката на данни представлява прехвърляне от текстови формат на записа, към формат удобен за използване за другите функции за изграждане на триизмерният модел на системата. Данните се прехвърлят в масив от структури от типа „data”.

Входните данни се четат от текстови файл (*.txt). Всеки ред представлява един запис с данни за един участък от канализационната система. Полетата в записа, описан в т. 2.3.2, са разделени едно от друго със символ за табулация ASCII 9 (TSV формат). Функцията „ReadingData()” чете входния файл ред по ред. Всеки ред се обработва, като според броя на прочетените до момента символи за табулации се определя поредният номер на полето от записа в структурата „data”. В зависимост от типа на данните за съответното поле се прави подходяща обработка на съответната част от текста в прочетения ред от файла, така че да се получат стойностите. Изключително важно е за работата с големи

масиви от данни е правилното заделяне и освобождаване на памет за тяхното съхранение.

```
lines = (data*)malloc(sizeof(data)); //Пример за първоначално заделяне на памет
                                     за един елемент от масива
....
lines = (data*)realloc(lines,(num_lines+1)*sizeof(data)); //Пример за реалокиране
                                                          на памет за масива „lines"
```

Съществен момент в работата на функцията е създаването на локална координатна система с отправна точка геометричния център на канализационната мрежа. Центърът на локалната координатна система се дефинира с отместване по трите координатни оси (offset) спрямо глобалната координатна система, в който формат са зададени входните данни. Отместването, като стойности се запазва с цел правилното извеждане на данните.

Определянето на геометричният център на канализационната мрежа става чрез намирането на граничните координати по съответните оси и определянето на тяхната среда.

```
long int offset_x=(min_x+max_x)/2;
long int offset_y=(min_y+max_y)/2;
long int offset_z=(min_z+max_z)/2;
```

Функцията завършва своето действие със затварянето на входния файл и освобождаването паметта на работните променливи.

Файлът „menu.h" съдържа в себе си функции и класове за изграждането на менюта, бутони и други елементи свързани с потребителския интерфейс.

Един от основните елементи на потребителският интерфейс , това са бутоните. Един бутон може да има три основни състояния:

- Достъпен
- Избран
- Натиснат

Достъпен е този бутон , който може да бъде избран и натиснат.

Избран е този, върху който се намира курсора на мишката.

Натиснат е този, който е избран и е натиснат левия бутон на мишката.

Това е реализирано чрез класа „button”, който съдържа в себе си методи за контрол над състоянията на бутона, достъп и манипулация на позиция и размер, както и проверка за намеса от потребителят.

```
class button{
  float x,y; //позиция на екрана
  float height, width; //височина и широчина
  bool slct,enbl,prss; //флагове за състояния избран, достъпен, натиснат

public:
...
```

Класът бутон съдържа конструктор, който позволява директно да се зададат параметрите на бутона още при дефинирането на нов обект.

Класа бутон има няколко основни метода:

bool selected() – проверява дали бутона е „избран”

void select() – задава състояние на бутона „избран”

void deselect() – премахва състояние на бутона „избран”

bool enabled() – проверява дали бутона е „достъпен”

void enable() – задава състояние на бутона „достъпен”

void disable() – премахва състояние на бутона „достъпен”

bool pressed() – проверява дали бутона е в състояние „натиснат”

int detect(float mouse_x,float mouse_y,bool click) – приема като параметри местоположението на курсора на мишката върху екрана и флаг за натиснат ляв бутон. Метода връща 0 , 1 или 2 в зависимост от различните случаи. Връща 0 , когато бутона е в състояние „достъпен” и курсора на мишката не се намира върху бутона. Връща 1, когато бутона е в състояние „достъпен” и курсора на мишката е върху него, а бутона преминава в състояние „избран”. Връща 2, когато бутона е в състояние „достъпен”, курсора на мишката е върху бутона и левия бутон е натиснат. Тогава бутона преминава в състояние „натиснат”.

Във файла main.cpp се съдържа основният алгоритъм на приложението. Във него са дефинирани някои основни функции свързани с потребителския интерфейс, правилното записване на данните в подходящите масиви от структури, инициализацията на OpenGL и извеждането на данните за триизмерния модел на канализационната система във външен файл.

Файлът main.cpp съдържа функцията BuildSewage() , която се грижи за това данните да бъдат правилно разпределени към масивите от обекти тръби и шахти.

```
void BuildSewage(int num_pipes,int &num_sh,pipe* &pipes,shaft* &shafts,data*
lines,int &progress){
      ....
   for(int i=0; i<num_pipes; i++){
         pipes[i].build_pipe(lines[i].name_branch,
                             lines[i].tube_inner_d,
                             lines[i].tube_outer_d,
                             lines[i].hor_length,
```

```
                                lines[i].slope,
                                lines[i].start_x,
                                lines[i].start_y,
                                lines[i].start_z,
                                lines[i].end_x,
                                lines[i].end_y,
                                lines[i].end_z,
                                100);

            if(lines[i].tube_inner_d>1500) temp_d=3000;
            else if(lines[i].tube_inner_d>1000) temp_d=2000;
            else if(lines[i].tube_inner_d>600) temp_d=1500;
            else temp_d=1000;

            if(num_sh>0){
                  ….. //Ако има повече от една шахта се прави проверка за
                        дублиране на шахти и задаване на параметри според ясни
                        критерии.
            else{
                  ….. //Ако няма шахти, се взима първата шахта от първия участък.
            }
            progress=i+1; //Увеличаване на прогреса
      }
}
```

Като параметри се подава броят на тръбите, получен при прочитането на файла, броят на шахтите, масив от обекти „pipes" (тръби), масив от обекти „shafts" (шахти), масив от структура тип dataa, и прогрес от работата на функцията. Броят на тръбите съответства на броят редовете на входния файл. Броят на шахтите, масивите от тръби и шахти се записват в резултат от работата на разглежданата функция.

В една шахта могат да се свързват една или повече тръби. Дълбочината на шахтата се определя от тръбата с най ниската точка на свръзка. Диаметърът на шахтата зависи от тръбата с най-голям диаметър.

Както се вижда на примерният код, функцията се състои от един основен цикъл , който работи, докато не се стигне до края на всеки запис. Всяко поле от всеки запис се подава като параметър за построяването на тръбите. В променливата „temp_d” се записва временно за конкретния запис, какъв е диаметъра на шахтите в края на участъка в зависимост от диаметъра на тръбата. Критериите за избор на диаметър на шахтата са описани във втора глава в т. 2.3.2.

Когато се обработва първият запис, тогава броят на шахтите е равен на нула. За да не се извършва проверка за дублиране , се взема началната шахта на първия участък и се увеличава броя на шахтите.

Проверката за дублиране се извършва, като чрез оператора за сравнение на класа „pipe”. Ако се намерят дублирани шахти се взима по-големият диаметър от двата и по-ниското дъно.

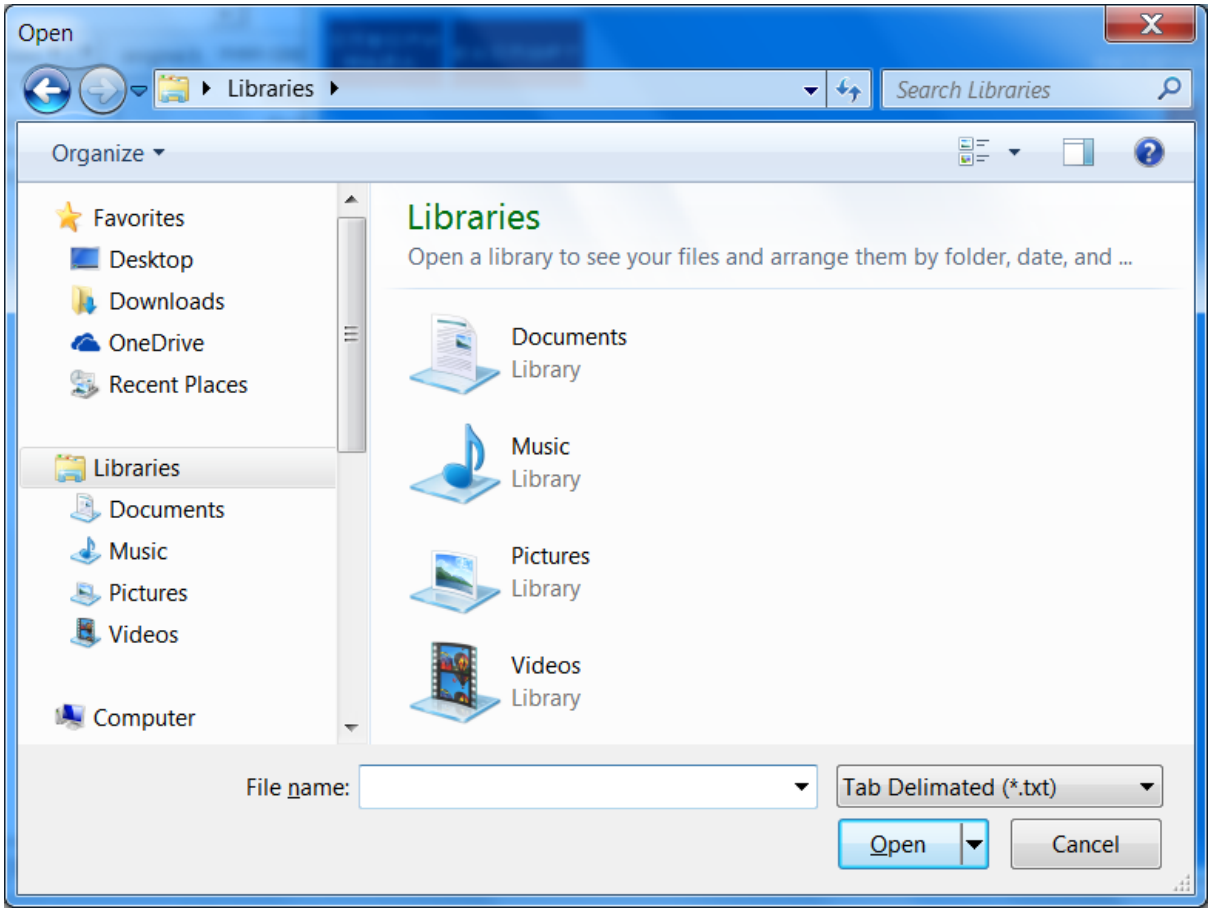
За да може потребителят да отваря входните файлове с данни е нужно да се предостави прозорец, от който да се избере входен файл и да има възможност за навигация във файловата система.

За това се грижи функцията

```
data* OpenFile(int &num_pipes,int &progress)
```

Тя връща масив от структури от типа data, в които се съдържат структурираните данни от входният файл. Като параметри се задават брой на тръби и променлива за резултат от прогрес на функцията.

Работата на функцията се състои в това да извика функция към операционната система Windows да отвори диалогов прозорец за избор на файл за отваряне. Този прозорец би изглеждал така , както е показано на фиг. 3.8.



Фиг. 3.8

Когато потребителят натисне бутона „Open” (при избран вече файл), прозорецът се затваря и функцията изпратена към ОС връща адреса на входния файл във файловата система. Когато потребителят натисне бутона „Cancel”, се връща празен низ от символи.

Функцията OpenFile() проверява, дали стрингът от диалоговия прозорец е празен. Ако е така , тогава тя прекратява своето

действие и връща празен масив с данни и брой на тръби равен на 0. Ако всичко е наред, функцията извиква в себе си функцията ReadingData , като подава името на файла за обработка и съответните параметри в които се записват данните.

Накрая функцията връща масива от данни и ако всичко е наред променливата за прогрес, трябва да е равна на броят на тръбите.

Според изискванията за генериране на външен файл, описани в т. 2.1.5 , трябва да се предостави възможност на потребителят да експортва файлове в стандартен формат. За целта е избран OBJ Wavefront 3D формат. Предимствата на този формат са, че той е широко използван от много видове софтуер за 3D графика, CAD, CAM системи, както и 3D принтери. Той е добре структуриран и не е нужна специална обработка за прочитането му [3].

За това се грижи функцията ExportFile(int num_pipes,int num_sh,pipe* pipes, shaft* shafts)

Тя приема, като параметри броят на тръбите, броят на шахтите, масив от обекти „тръби”, масив от обекти „шахти”. Така чрез приетите параметри , данните от моделите на тръбите и шахтите се записват последователно във избран файл от потребителя.

При повикването на тази функция се отваря диалогов прозорец, който позволява навигация на потребителя из файловата система и избор на изходен файл. Този прозорец изглежда аналогично на този показан на фиг. 3.8. При натискане на бутона „Save” се връща низ от символи указващ адреса на изходния файл.

3.2 Реализация на алгоритъма

Чрез функцията WinMain() се реализира главният алгоритъм на приложението. Тъй като обемът на тази функция е голям, ще бъдат описани основните операции, които се извършват, а те са:

- Деклариране на обект „прозорец”
- Деклариране на обекти от тип button за UI
- Задаване на стойности на параметрите на обекта „прозорец”
- Създаване на прозорец
- Стартиране на втората програмна нишка
- Първоначално установяване на OpenGL
- Зареждане на текстури
- Главен цикъл (работи, докато не е подадена команда за изход)
    - Следене на позицията на курсора на мишката и състоянието на нейните бутони
    - Следене на клавиатура
    - Обработка на състоянието на екранните бутони
    - Действия по преоразмеряване на екранния прозорец
    - Управление на ъгъла на гледната точка (камерата) чрез прехвърлянето на екранните координати на курсора на мишката в полярни.
    - Задаване на фонов цвят на прозореца
    - Дефиниране на източник за светлина
    - Задаване на позицията на камерата
    - Задаване на мащаб на модела
    - Изчертаване на модела
    - Информационни надписи над елементите на модела
    - Изчертаване на компас
    - Информационни надписи за обратна връзка към потребителя
    - Изчертаване на бутоните на UI
    - Смяна на изображението от буфера за изчертаване на екранната памет.
- Деинициализация на OpenGL
- Унищожение на програмния прозорец
- Край на програмата

Алгоритмът за работа на втората нишка, който е описан в т.2.3.1 е изграден със следния код:

```
void Phys_Thread(void* PARAMS){
   while(1){
      Sleep(10);
      //Do something

      if(o_cmd){
                opened_file=false;
               spec.set_coords(0,-400,100);
               global_scale=0.1;
               num_pipes=0;
               num_sh=0;
               if(num_pipes){
               free(lines);
               free(pipes);
               free(shafts);
               }
               lines = OpenFile(num_pipes,progress);
               if(num_pipes){
                  opened_file=true;
                  BuildSewage(num_pipes,num_sh,pipes,shafts,lines,progress);
               }
               else opened_file=false;
               spec.set_coords(0,0,0);
               o_cmd=0;
               any_btn=false;
      }
      if(e_cmd){
         ExportFile(num_pipes,num_sh,pipes,shafts,shaft_sd);
         e_cmd=0;
      }

      if(move_forward){
         spec.move_forward(cam_speed);
      }
      if(move_back){
         spec.move_back(cam_speed);
      }
      if(move_left){
         spec.move_left(cam_speed);
      }
      if(move_right){
         spec.move_right(cam_speed);
      }
      if(move_up){
         spec.move_up(cam_speed);
      }
      if(move_down){
         spec.move_down(cam_speed);
      }
   }
}
```

Четвърта глава

Ръководство за потребителя

4.1 Системни изисквания.

Конкретното приложение няма високи изисквания, което позволява използването му и на по-стар хардуер. Системните изисквания са определени след множество тестове на различни компютри.

Минимални изисквания:

Процесор: Intel Celeron/AMD 600MHz
Оперативна памет (RAM) 256MB
Видео карта: Инсталиран драйвер за OpenGL с версия 3.0 или по-висока,
Video RAM: 32MB
Свободно място на хард диск: 4MB

Препоръчителни изисквания:

Процесор: Intel Celeron/AMD 1.6GHz
Оперативна памет (RAM) 512MB
Видео карта: Инсталиран драйвер за OpenGL с версия 3.0 или по-висока,
Video RAM: 128MB
Свободно място на хард диск: 8MB

Изисквания към операционна система:

Windows XP или по-нова версия.

Резултати от направените тестове, показват че приложението се държи стабилно на многоядрени процесори, като споделя ресурсите на приложението между ядрата. Тествано е на процесори, като AMD Athlon, AMD A6, Intel i3, Intel i5, Intel Celeron и др. Приложението изисква задължително инсталиран драйвер за OpenGL минимум версия 3.0. Ако няма никакъв инсталиран драйвер,

това може да доведе до влошаване на изобразената картина и до неравномерна работа на приложението. Приложението е тествано на различни видео карти, от семейството на ATI/AMD Radeon и Nvidia Gforce, като показва много добри резултати при повече от 128MB VRAM.

В зависимост от вашата компютърна конфигурация, кадрите в секунда на приложението могат да се колебаят между 24 и 500. Понякога горната граница е ограничена от самата видеокарта до 50 или 60 кадъра в секунда.

4.2 Инсталация

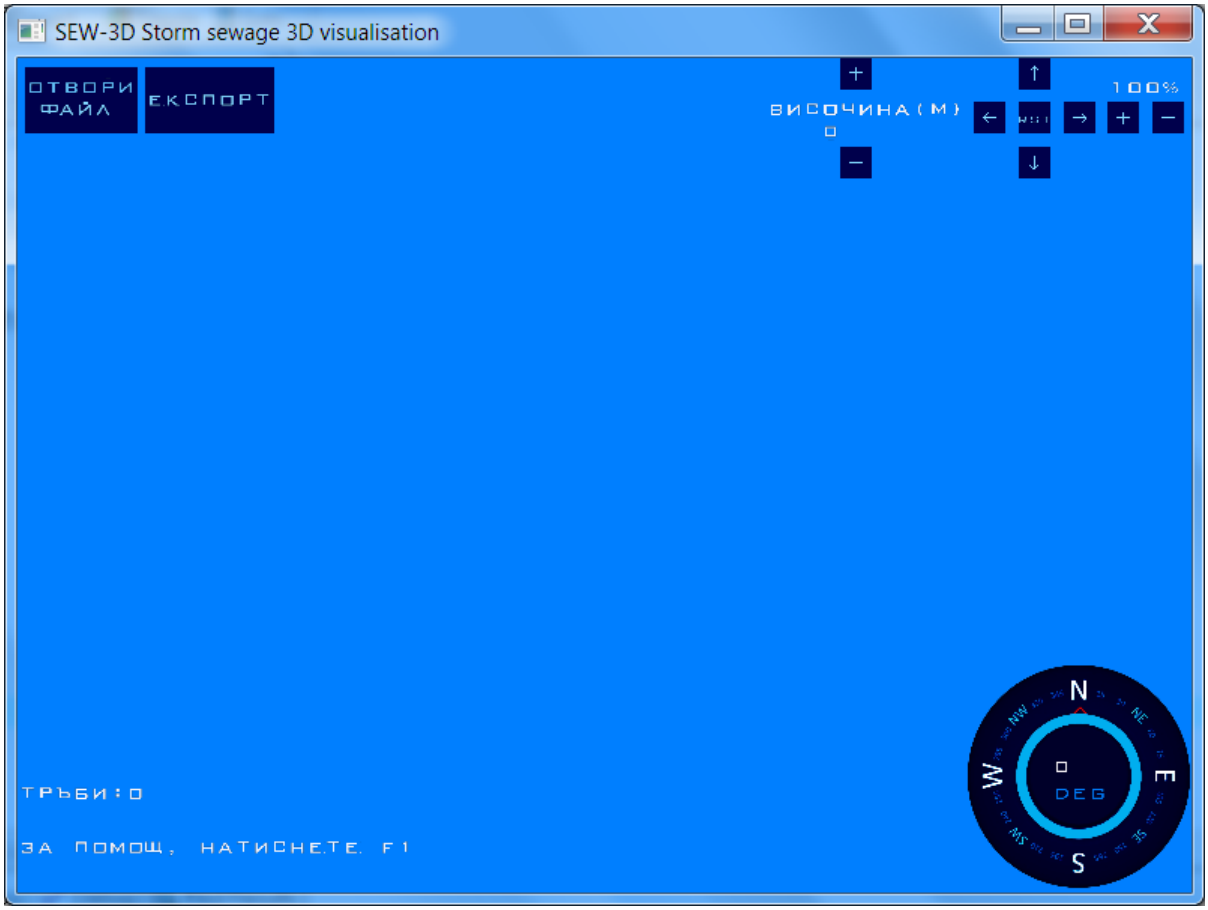
Приложението не изисква специална инсталация. Директорията на приложението може да бъде копирана в която директория искате.

Приложението може да се стартира и пренася от външен хард диск, флаш памет, карти памет и др. файлови записващи устройства.

4.3 Начин на работа с приложението

Приложението се стартира от файла „SEW-3D.exe”.

На фиг. 4.1 е показано как изглежда екрана на приложението. Разполага с два бутона в горния ляв ъгъл за работа с файлове, бутони за навигация в горен десния ъгъл, компас за ориентация в долен десен ъгъл и данни за броят на тръбите, упътване за помощ в долния ляв ъгъл.

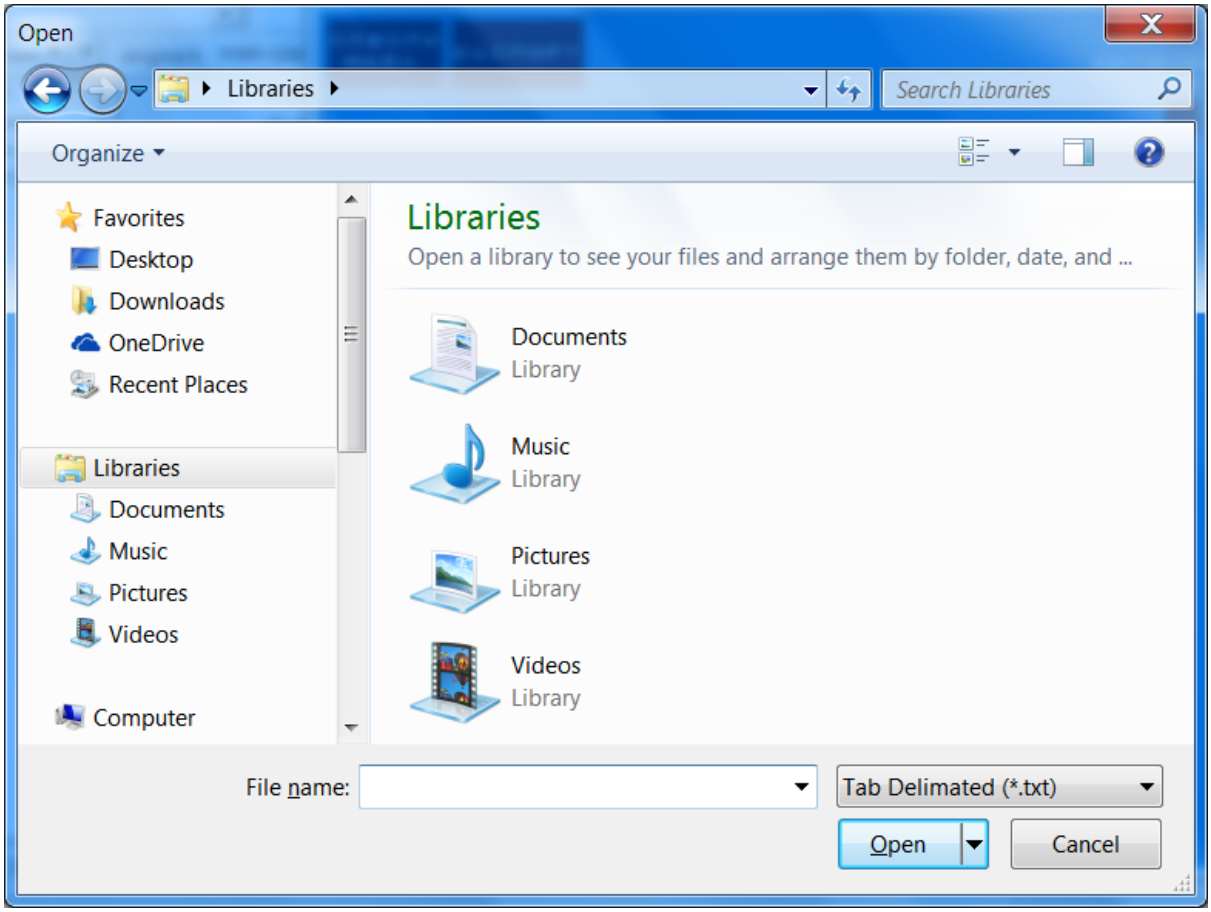


Фиг. 4.1

Входните данни се подават във формата описан в т. 2.3.2, като полетата са разделени със символ за табулация (ASCII 9).

4.3.1 Отваряне на файл

За да отворите файл, натиснете бутона с надпис „Отвори файл” или бързата клавишна комбинация [Ctrl] + [O]. Ще се отвори диалогов прозорец, чрез който можете да изберете вашия файл (фиг. 4.2).

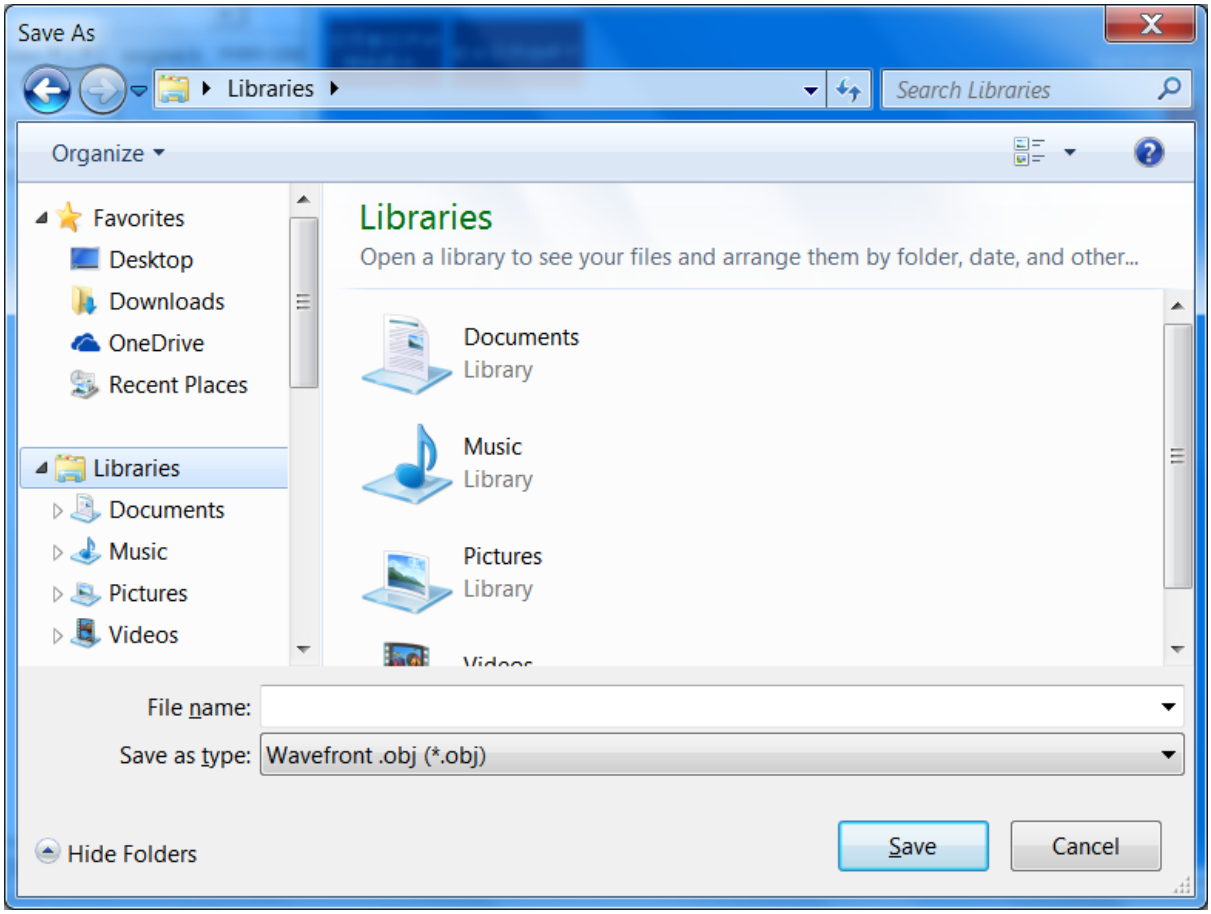


Фиг. 4.2

Прочитането на файла може да отнеме известно време в зависимост от големината му.

4.3.2 Експорт на файл

За да запазите 3D модела на канализационната мрежа във „*.obj" формат е нужно да натиснете бутона с надпис „Експорт" или бързата клавишна комбинация [Ctrl] + [E]. Аналогично като при отварянето на файл, ще излезе диалогов прозорец, който ще ви предостави възможност да изберете къде да запишете вашия изходен файл (фиг. 4.3).



Фиг. 4.3

Процеса на експорт може да отнеме известно време, затова е препоръчително да изчакате търпеливо, докато операцията завърши.

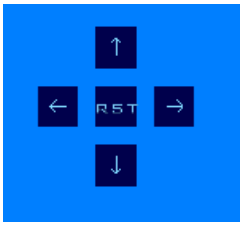
4.3.3 Навигация.

4.3.3.1 Задаване на ъгъл на камерата

Уверете се че курсора на мишката не е върху някой от бутоните. Натиснете левия бутон и започнете да я движите. Движението нагоре и надолу на мишката променя изгледа на камерата да гледа нагоре или надолу. Движението наляво или надясно, определя ъгъла под който е завъртяна камерата спрямо север.

4.3.3.2 Хоризонтално движение

Движението се извършва спрямо посоката на гледната точка. Има бутони в горния десен ъгъл с иконки на стрелки, с посоки напред, назад , наляво , надясно (Фиг. 4.4). При натискането им се извършва движение в съответните посоки. Движението може да се осъществи и чрез натискане на бутоните стрелки от клавиатурата. При натискането на бутона с надпис „RST”, позицията на гледната точка се връща в центъра на координатната система.



Фиг.4.4

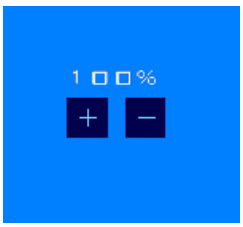
4.3.3.2 Промяна на местоположението на гледната точка по надморска височина

Покачването или снижаването на височината на гледната точка може да стане чрез натискане на бутоните с иконка плюс и минус, разположени в ляво от бутоните за движение на Фиг. 4.4. Също така височината може да се променя чрез бутоните [Shift] и [Space] Между бутоните за промяна на височината е разположен текст с обратна връзка към потребителят на каква надморска височина се намира гледната точка (Фиг. 4.5).

Фиг. 4.5

4.3.3.3 Увеличаване и намаляване на мащаба на модела.

Чрез бутоните разположени в дясно от бутоните за навигация , можете да променяте мащаба на модела. Бутона с иконка плюс увеличава мащаба а този с иконка минус го намалява. Над тях е разположен надпис, който показва сегашния мащаб на модела (фиг. 4.6) Същата функцията извършват бързите клавишни комбинации [Ctrl] + [+] и [Ctrl] + [-].



Фиг. 4.6

Заключение

Разработеният софтуер за 3D визуализация на селищни канализационни мрежи покрива всички изисквания, дадени по задание и описани в т.2.1

Направени са тестове с реални данни от съществуващи проекти, както и с данни от заснемане на съществуващи канализационни мрежи. Генерираните с този софтуер триизмерни модели съответстват вярно с входните данни. При разглеждането на триизмерния модел на проектно решение , бяха забелязани грешки, допуснати при проектирането. Откриването на тези грешки само по табличните данни би отнело значително време на проектанта. Работата на програмата бе посрещната с одобрение от специалисти в областта на строителното инженерство.

Настоящият софтуер е иновация и ще търпи своето развитие. Освен нормалните подобрения във функционалността и производителността се предвижда засилване на симулационната част. Добавянето в триизмерния модел и други подземни мрежи от селищната инфраструктура (водопровод, газопровод, електрозахранване, съобщителни мрежи и др.) ще даде възможност да се визуализира пълната картина на взаимното местоположение на тези мрежи и избягването на конфликтни точки.

Използвана литература

1. C++ - http://www.cplusplus.com/
2. OpenGL - F.S Hill, Computer Graphics Using OpenGl 2nd Edition, Prentice Hall, 2001
3. Wavefront Obj - http://en.wikipedia.org/wiki/Wavefront_.obj_file

Съдържание