МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук

Кафедра інформаційних технологій

ДИПЛОМНА РОБОТА

Рівень вищої освіти бакалавр

на тему: Проектування графічних баз даних у відкритому ПЗ BRL-CAD

Виконав студент 4 курсу групи К-41
Напрям підготовки 6.050101
комп'ютерні науки
Мазур Олег Анатолійович

Керівник к. т. н., доцент
Великодний Станіслав Сергійович

Консультант _____
_____

Рецензент к. ф.-м. н., доцент
Буяджи Василь Володимирович

Одеса 2018

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ ВИМІРЮВАННЯ, СКОРОЧЕНЬ І ТЕРМІНІВ

BRL – Ballistic Research Laboratory
CSG – constructive solid geometry
DBTG – Data Base Task Group
GNU – General Public License
MGED – Multiple-Device Geometry Editor
NURESIM – NUclear REactor SIMulations
RT – Ray Tracer
SDC – System Development Corporation
VLDB – Very Large Database
БД – база даних
ГБД – графічна база даних
ГВ – графічне вікно
ДА – діаграма активності
ДВВ – діаграма варіантів використання
ДК – діаграма класів
ДКМ – діаграма компонентів
ДП – діаграма послідовності
ДР – діаграма розгортання
ДС – діаграма станів
ЕОМ – електронна обчислювальна машина
ІПС – інформаційно-пошукова система
ІС – інтегральна схема
КВ – командне вікно
КД – комунікативна діаграма
НВІС – надвелика інтегральна схема
ПЗ – програмне забезпечення
ПК – персональний комп’ютер
СУБД – система управління базами даних

ВСТУП

В наш час є велика кількість програмних засобів, які виконують велику кількість спеціалізованих задач. Деякі з них зав’язані лише на якусь одну галузь промисловості, інші – застосовуються у великій кількості, але тенденція йде шляхом спеціалізації програмних продуктів в цілому.

Корпорації, що розроблюють системи автоматизованого проектування (САПР) – проектують багато спеціалізованих програмних продуктів, наприклад – AutoDesk. В них є цілий комплект програм для роботи с машинобудівним профілем (Inventor), архітектурою (ArchiCad), дизайном (3dMAx) та проектуванням у широкому сенсі (AutoCad).

Згідно із загальноприйнятою методологією – САПР класифіковано за трьома рівнями:

– прості – це більша частина програмних продуктів виробника Autodesk, ADEM, КОМПАС та інші;

– середні – програмні продукти рівня SolidWorks;

– високі – спеціалізовані програмні продукти високої потужності (CA-TIA).

В цілому – САПР – комплекс засобів автоматизації проектування, що взаємопов'язано із підрозділами проектної організації, яка виконує автоматизоване проектування. Під автоматизацією проектування розуміють такий спосіб проектування, при якому всі проектні операції, процедури або їх частини здійснюється при взаємодії людини та електронної обчислювальної машини (ЕОМ).

В результаті функціонування САПР – від технічного завдання, послідовно, проходячи низку проектних стадій, замовник одержує робочий проект об'єкта проектування (робочі креслення, технічний опис та ін.).

САПР виконує низку процедур, певним чином тих, що логічно пов'язані між собою і виконавцями для прийняття проектних рішень. САПР можуть бути використані в різних галузях науки, техніки й виробництва: для автоматизації проектування окремих деталей, предметів та вузлів; механізмів і машин, комплексів машин і агрегатів; виробничих ліній цілих виробничих підприємств і комплексів.

Розробка, впровадження і розвиток елементів і підсистем САПР в різних підгалузях текстильної й легкої промисловості – є однією з важних задач з виведення галузі на сучасний рівень світового промислового виробництва товарів широкого вжитку. Завдяки потужним обчислювальним засобам у САПР за допомогою інформаційно-пошукових систем (ІПС), що містять банки і бази даних (БД) готових проектно-конструкторських рішень,

можливо досить швидко внести зміни в розміри, форму та порядок обробки деталі, що виготовляється, у послідовність технологічних операцій, переорієнтувати увесь виробничий процес.

Розроблені елементи і підсистеми САПР дають значну ефективність в тому випадку, коли результати автоматизованого проектування використовуються у звичайному виробництві. Ефективність, в даному випадку, обумовлюється тим, що за сучасних темпах розвитку науки і техніки, виникає протиріччя між зростаючим рівнем науково-технічних досягнень та існуючими методами й засобами традиційного проектування.

Процес проектування нового складного виробу триває, при встановленому порядку, кілька років. За цей час, у ряді галузей з'являються нові наукові ідеї та рішення, які виводять виробництва на новий рівень і породжують нове покоління машин, приладів та установок. Розробка та впровадження елементів і підсистем САПР дозволяє, значною мірою, подолати протиріччя між темпами розвитку науки і техніки та процесів проектування, підвищити ефективність проектування, скоротити його терміни. Використання САПР дає значне підвищення продуктивності праці в усіх сферах виробництва.

У методологічному плані, САПР можна визначити як комплекс наукових методів дослідження діяльності в галузі проектування з метою розробки системи методик, математичних моделей і алгоритмів, що забезпечують найкращий розподіл функції між проектувальником і машиною, при пошуку оптимальних проектних рішень. У результаті вивчення об'єктів проектування, розробляються моделі та алгоритми, які з максимально можливою точністю фіксують закономірності формоутворення об'єктів та логіки у процесах їх проектування.

Відповідно до принципу «людина-машина», що покладено в основу побудови САПР, в технічному аспекті, систему можна визначити як комплекс інформаційного, програмного, технічного забезпечення системи підпорядкованої єдиної методології машинного проектування. Технічні пристрої, багато в чому, визначають ефективність систем, що створюються, тому їм надається особливе значення.

Підсистеми САПР визначають зону діяльності в системі проектування з вирішенням всіх завдань, що пов'язані з розробкою проектів для кожної з виділених груп об'єктів. Кожна підсистема САПР поєднує в собі діяльність з проектування всіх об'єктів одного виду та відносяться до однієї певної області проектування. У свою чергу, підсистеми САПР мають свій поділ на субпідсистеми, тобто на взаємопов'язані зони, що спеціалізовані за проектуванням об'єктів різного призначення. Межі між підсистемами САПР

відзначають замовлення на проектування (вхід в підсистему) від завдання на проектування, яке є початковою стадією в проектному процесі і в результаті якої може народжуватись замовлення на проектування об'єкта в іншій підсистемі.

Проте проектування та звітні креслення – це тільки одна сторона САПР. Другою, не менш важливою частиною проектування – є моделювання. З розвитком САПР почали уходити з практики макетування моделей та тестування у реальності. Нішу практичного випробування зайняло комп'ютерне моделювання – це стало великим проривом тому, що у комп'ютерній системі можливо спроектувати деталь з будь-якого матеріалу та набагато легше зробити чисельний аналіз показників. Також було розв'язано проблему середи, у якій проводяться випробування, тепер її можливо спроектувати будь-якою.

Однією з важливих складових частин САПР – є *машинна графіка*, як сукупність засобів та прийомів, за допомогою яких здійснюється введення, перетворення та виведення з ЕОМ графічної інформації. Машинна графіка – нова *актуальна* галузь проектування та застосування засобів обчислювальної техніки, що інтенсивно розвивається у останній час.

Термін «машинна графіка» означає обробку на ЕОМ графічної інформації, а також виведення результатів у вигляді різних графічних зображень. Графічна інформація – найбільш ємне і наочне уявлення великого обсягу інформації, однак, практичне застосування машинної графіки довгий час стримувалось відсутністю відповідного обладнання та математичного забезпечення.

Особливий інтерес до машинної графіки став виявлятись у зв'язку із розвитком автоматизованих систем проектування на базі ЕОМ, що інтенсивно розроблюються і впроваджуються у даний час не тільки в машинобудуванні, приладобудуванні, радіоелектроніці, але і в інших галузях.

Відмінною складовою завдань машинної графіки – є обробка графічних баз даних (ГБД), які, по суті являють собою «звичайні» БД, але в основу яких закладено математичні алгоритми відновлення зображення за сформованими статистичними координаційними даними. Такі можливості є далеко не у кожній САПР, але сучасні тенденції просто вимагають цього. Велика кількість програмних продуктів розроблюється з широким спектром моделюючих характеристик, BRL-CAD – це одна з таких САПР.

BRL-CAD – це спеціалізована кросплатформова система з відкритим кодом. Вона являє собою потужну 3D САПР для моделювання об`ємних тіл *методами* CSG. Ця САПР включає в себе інтерактивний геометричний

редактор, паралельне трасування променів, рендерінг та геометричний аналіз.

BRL-CAD розроблявся понад 20 років та набув широкого застосування у збройних силах США. Увесь BRL-проект працює із вихідного коду, а тому його можна використовувати на будь-яких платформах: GNU/Linux, MacOS, Solaris та Windows.

Мета поданої дипломної роботи полягає у створенні, підключенні та оптимізації графічних баз даних, як композиційного компоненту відкритої САПР BRL-CAD.

Об'єктом роботи – є відкриті ГБД, що редагуються засобами BRL-CAD.

Пояснювальна записка до дипломної роботи містить: ____ сторінок, ___ рисунків, ___ таблиць та ___ використаних джерел.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Проблема визначення

У літературі пропонується безліч визначень поняття «база даних», що відображають скоріше суб'єктивну думка тих чи інших авторів, однак загальновизнане єдина формулювання відсутнє.

Основним визначенням для БД, яке використовується у питаннях визначення інтелектуальної власності, – є визначення, що надається статтею 420 цивільного кодексу України (затв. наказом №435 від 16.01.2003): база даних – це представлена в об'єктивній формі сукупність самостійних матеріалів (статей, розрахунків, нормативних актів, судових рішень та інших подібних матеріалів), систематизованих таким чином, щоб ці матеріали могли бути знайдені і оброблені за допомогою електронної обчислювальної машини (ЕОМ) [1].

Проте, як ми бачимо це визначення містить яскраво визначений юридичний акцент, тому набуло більш адаптоване для користувачів БД визначення, сформульоване у Законі України «Про авторське право і суміжні права». База даних (компіляція даних) – сукупність творів, даних або будь-якої іншої незалежної інформації у довільній формі, в тому числі – електронній, підбір і розташування складових частин якої та її упорядкування – є результатом творчої праці, і складові частини якої – є доступними індивідуально і можуть бути знайдені за допомогою спеціальної пошукової системи на основі електронних засобів (комп'ютера) чи інших засобів [2].

Слід зазначити, що багато фахівців вказують на поширену помилку, що складається в некоректному використанні терміна «база даних» замість терміна «система управління базами даних» (СУБД), і вказують на необхідність розрізнення цих понять [3].

Одночасно з цим, вживаються й визначення БД з міжнародних стандартів:

База даних – сукупність даних, що зберігаються у відповідності зі схемою даних, маніпулювання якими виконують відповідно до правил засобів моделювання даних [4].

База даних – сукупність даних, організованих відповідно до концептуальної структури, яка описує характеристики цих даних і взаємовідносини між ними, причому таке зібрання даних, яке підтримує одну або більше областей застосування [5].

Приклад такої концептуальної структури із визначеними зв’язками між даними подано на рис. 1.1.

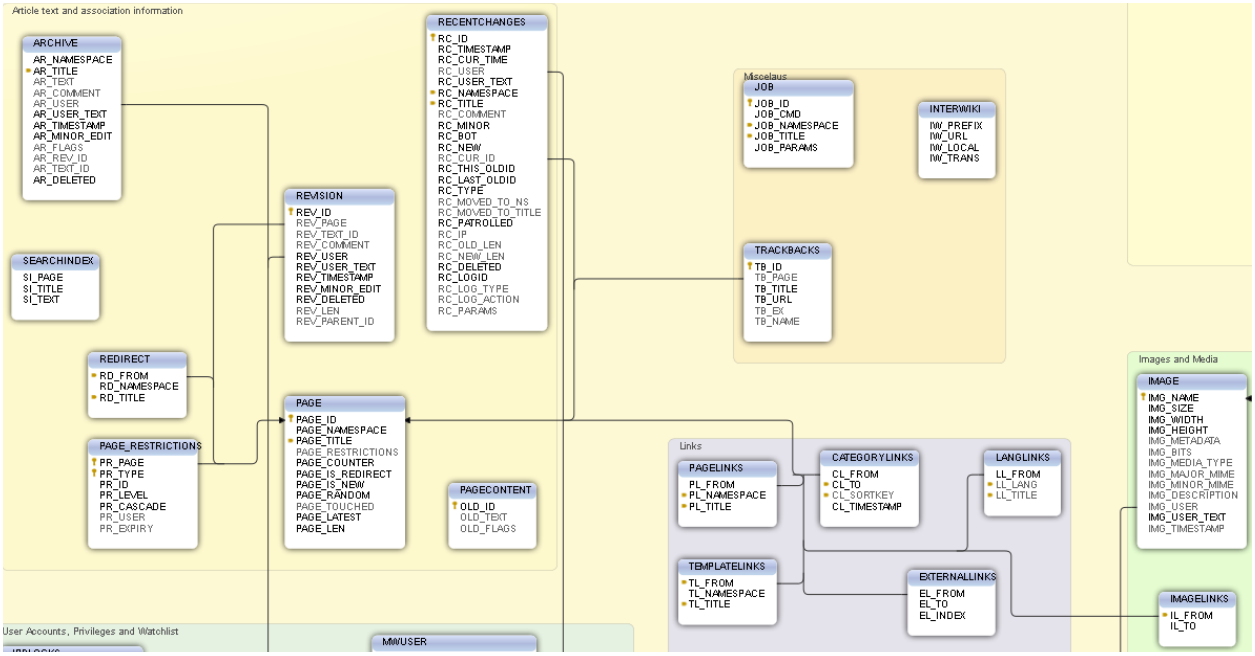

Рисунок 1.1 – Типова діаграма структури БД

Нижче приведемо визначення БД з декількох авторитетних монографій:

База даних – організована відповідно до певних правил і підтримувана в пам'яті комп'ютера сукупність даних, що характеризує актуальний стан деякої предметної області і використовувана для задоволення інформаційних потреб користувачів [6].

База даних – деякий набір перманентних (постійно збережених) даних, що використовуються прикладними програмними системами підприємства [7].

База даних – спільно використовуваний набір логічно зв'язаних даних (та опис цих даних), призначений для задоволення інформаційних потреб організації [8].

При аналізі усіх визначень БД (додавши ще й [9]), слід помітити найбільш часто (явно або неявно) присутні наступні відмітні ознаки:

а) БД зберігається і обробляється у обчислювальній системі (таким чином, будь-які позакомп’ютерні сховища інформації (архіви, бібліотеки, картотеки та ін.) БД не є);

б) дані в БД логічно структуровані (систематизовані) з метою забезпечення можливості їх ефективного пошуку та обробки в обчислювальній системі (структурованість передбачає явне виділення складових частин (елементів), зв'язків між ними, а також типізацію елементів і зв'язків, за якої з типом елемента (зв'язком) співвідноситься певна семантика й припустимі операції);

в) БД включає схему, або метадані, що описують логічну структуру БД у формальному вигляді (відповідно до деякої мета моделі, наприклад, відповідно до [4]: «постійні дані в середовищі БД включають в себе схему і БД. Схема включає в себе описи змісту, структури і обмежень цілісності, використовувані для створення і підтримки БД. БД включає в себе набір постійних даних, визначених за допомогою схеми. Система управління даними використовує визначення даних у схемі для забезпечення доступу і управління доступом до даних у БД»).

З перерахованих ознак тільки перший – є строгим, а інші допускають різні трактування та різні ступені оцінки. Можна лише встановити деяку ступінь відповідності вимогам до БД.

У такій ситуації не останню роль відіграє загальноприйнята практика. Відповідно до неї, наприклад, не називають БД файлові архіви, Інтернет-портали або електронні таблиці, незважаючи на те, що вони, в деякій мірі, володіють ознаками БД. Прийнято вважати, що ця ступінь в більшості випадків недостатня (хоча можуть бути винятки).

1.2 Історія розвитку БД

Історія виникнення і розвитку технологій БД може розглядатися як у широкому, так і у вузькому аспекті.

У широкому аспекті поняття історії БД узагальнюється до історії будь-яких засобів, за допомогою яких людство зберігає й обробляє дані. У такому контексті згадуються, наприклад, засоби обліку царської скарбниці і податків в стародавньому Шумері (4000 р. до н.е.), вузликова писемність інків – кіпу, клинопису, що містять документи асирійського царства та ін. Слід пам'ятати, що недоліком цього підходу – є розмивання поняття «база даних» та фактичне його злиття з поняттями «архів» і навіть «писемність».

Історія БД у вузькому аспекті розглядає бази даних в традиційному (сучасному) розумінні. Ця історія починається з 1955 року, коли з'явилось програмоване обладнання обробки записів. Програмне забезпечення (ПЗ) цього часу підтримувало модель обробки записів на основі файлів. Для зберігання даних використовувалися перфокарти.

Оперативні мережеві БД з'явилися у середині 1960-х. Операції над оперативними БД оброблялися в інтерактивному режимі за допомогою терміналів. Прості індексно-послідовні організації записів швидко розвинулися до більш потужної моделі записів, орієнтованої на набори. За керівництво роботою Data Base Task Group (DBTG), наслідком виконання

якої стала розроблена стандартна мова опису і маніпулювання даними, Чарльз Бахман отримав Тьюрінгову премію [10].

В цей же час в співтоваристві БД COBOL була пророблена концепція схем БД і концепція незалежності даних.

Наступний важливий етап пов'язаний з появою на початку 1970-х реляційної моделі даних, завдяки роботам Едгара Ф. Кодда. Роботи Кодда відкрили шлях до тісного зв'язку прикладної технології БД з математикою і логікою. За свій внесок в теорію і практику Едгар Ф. Коддом також отримав премію Тюрінга [11].

Сам термін database (база даних) з'явився на початку 1960-х років, і був введений у вживання на симпозіумах, організованих фірмою SDC (System Development Corporation) у 1964 – 1965 роках, хоча розумівся спочатку в досить вузькому сенсі, в контексті систем штучного інтелекту. У широке вживання, у сучасному розумінні, термін увійшов лише в 1970-і роки [12].

1.3 Різновиди БД

Існує величезна кількість різновидів БД, що відрізняються за різними критеріями. Наприклад, у [6], визначаються понад 50 видів БД. Основні класифікації БД наведені нижче, а їх графічну структуру подано на рис. 1.2.

1.2.1 Класифікація за моделлю даних

Приклади класифікації за моделлю даних:

а) ієрархічна;

б) мережева;

г) реляційна;

д) об'єктна;

е) об'єктно-орієнтована;

ж) об'єктно-реляційна;

и) функціональна.

1.2.2 Класифікація за середовищем постійного зберігання

Приклади класифікації за середовищем постійного зберігання:

а) у вторинній пам'яті, або традиційна (англ. conventional database): середовищем постійного зберігання – є периферійна енергонезалежна пам'ять (вторинна пам'ять) – як правило жорсткий диск (в оперативну пам'ять СУБД поміщає лише кеш і дані для поточної обробки);

б) в оперативній пам'яті (англ. in-memory database, memory-resident database, main memory database): всі дані на стадії виконання знаходяться в оперативній пам'яті;

в) у третинній пам'яті (англ. tertiary database): середовищем постійного зберігання – є від'єднується від сервера пристрій масового зберігання (третинна пам'ять), як правило на основі магнітних стрічок (стримерів) або оптичних дисків (у вторинній пам'яті сервера зберігається лише каталог даних третинної пам'яті, файловий кеш і дані для поточної обробки; завантаження самих даних вимагає спеціальної процедури).

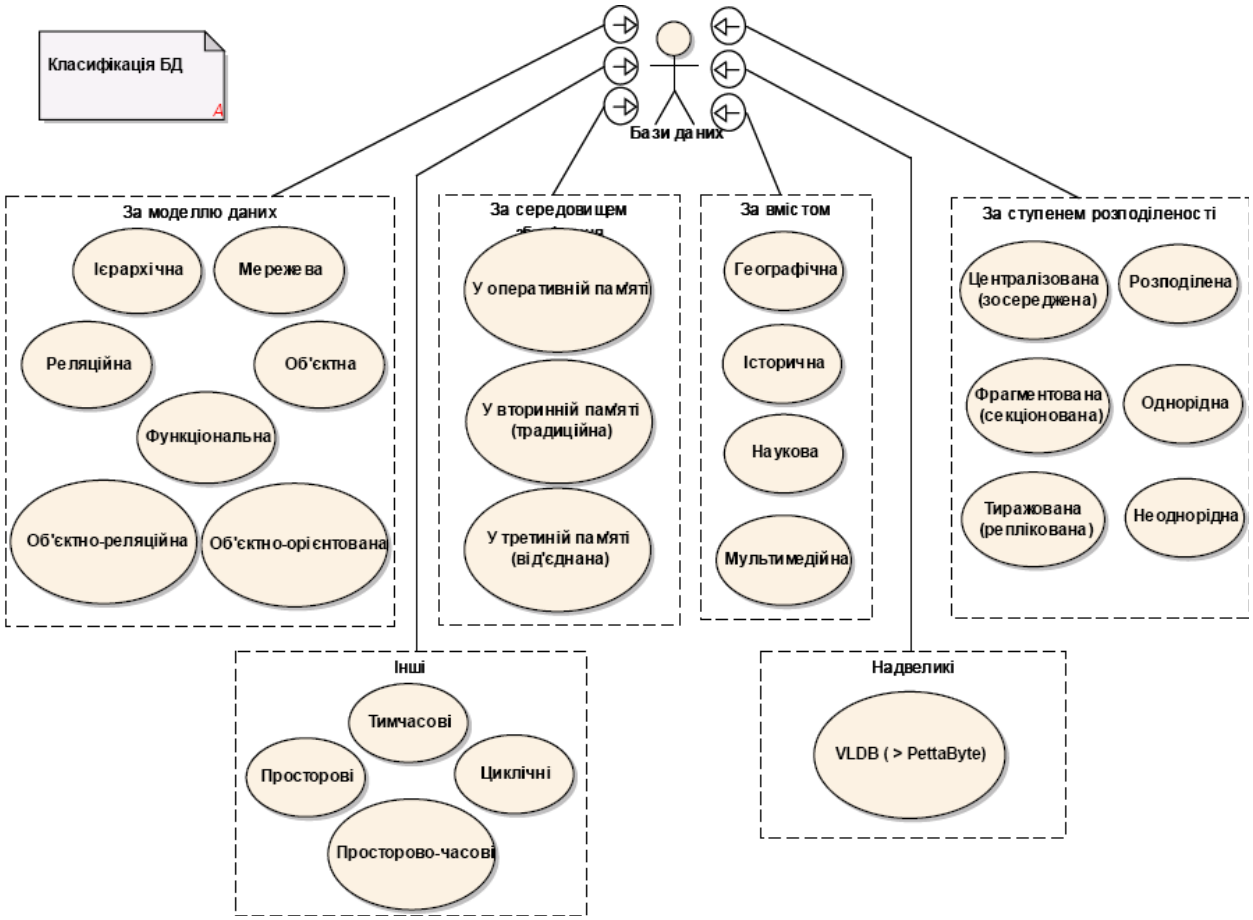

Рисунок 1.2 – Графічна структура принципу класифікації БД

1.2.3 Класифікація за вмістом

Приклади класифікації за вмістом:

а) географічна;

б) історична;

в) наукова;

г) мультимедійна.

1.2.4 Класифікація за ступенем розподіленості

Приклади класифікації за ступенем розподіленості:

а) централізована, або зосереджена (англ. centralized database): БД, що повністю підтримується на одному комп'ютері;

б) розподілена (англ. distributed database): БД, складові частини якої розміщуються в різних вузлах комп'ютерної мережі відповідно до будь-якого критерію;

в) неоднорідна (англ. heterogeneous distributed database): фрагменти розподіленої БД в різних вузлах мережі підтримуються засобами більше однієї СУБД;

г) однорідна (англ. homogeneous distributed database): фрагменти розподіленої БД в різних вузлах мережі підтримуються засобами однієї і тієї ж СУБД;

д) фрагментована, або секціонована (англ. partitioned database): методом розподілу даних – є фрагментованість (партіціонування, секціонування), вертикальне чи горизонтальне;

е) тиражована (англ. replicated database): методом розподілу даних – є тиражування (реплікація);

1.2.5 Інші види БД

Проте, не усі види БД можна класифікувати за вищенаведеними ознаками. Існують деякі БД, які традиційно відносяться до так званих інших видів БД, серед яких:

а) просторова (англ. spatial database): БД, в якій підтримуються просторові властивості сутностей предметної області,. такі БД широко використовуються в геоінформаційних системах;

б) тимчасова, чи темпоральна (англ. temporal database): БД, в якій підтримується будь-який аспект часу, не враховуючи часу, обумовленого користувачем;

в) просторово-часова (англ. spatial-temporal database) БД: БД, в якій одночасно підтримується одне або більше вимірів в аспектах як простору, так і часу;

г) циклічна (англ. round-robin database): БД, обсяг збережених даних якої не змінюється з часом, оскільки в процесі збереження даних одні і ті ж записи використовуються циклічно.

1.4 Надвеликі БД

Надвелика БД (англ. Very Large Database, VLDB) – це БД, яка займає надзвичайно великий обсяг на пристрої фізичного зберігання. Термін передбачає максимально можливі обсяги БД, які визначаються останніми досягненнями в технологіях фізичного зберігання даних і в технологіях програмного оперування даними.

Кількісне визначення поняття «надзвичайно великий обсяг» змінюється у часі; в даний час вважається, що це обсяг, вимірюваний щонайменше петабайтами. Для порівняння, в 2005 р. – найбільшими в світі вважалися БД з об'ємом сховища близько 100 терабайт [13].

Фахівці відзначають необхідність особливих підходів до проектування надвеликих БД. Для їх створення нерідко виконуються спеціальні проекти з метою пошуку таких системотехнічних рішень, які дозволили б хоч якось працювати з такими великими обсягами даних. Як правило необхідні спеціальні рішення для дискової підсистеми, спеціальні версії операційної середовища і спеціальні механізми звернення СУБД до даних [14].

Дослідження в галузі зберігання і обробки надвеликих баз даних VLDB завжди знаходяться на вістрі теорії і практики баз даних. Зокрема, з 1975 року проходить щорічна конференція International Conference on Very Large Data Bases («Міжнародна конференція з надвеликих баз даних»). Більшість досліджень проводиться під егідою некомерційної організації VLDB Endowment (Фонд цільового капіталу «VLDB»), яка забезпечує просування наукових робіт та обмін інформацією в галузі надвеликих БД і суміжних областях, зокрема ГБД.

1.5 Комп’ютерне моделювання складних графічних об’єктів

Комп'ютерне моделювання – є одним з ефективних методів вивчення складних систем. Комп'ютерні моделі простіше і зручніше досліджувати у силу їх можливості проводити так звані. обчислювальні експерименти, у тих

випадках, коли реальні експерименти ускладнені через фінансові або фізичні перешкоди або можуть дати непередбачуваний результат.

Логічність і формалізованість комп'ютерних моделей дозволяє виявити основні фактори, що визначають властивості досліджуваного об'єкта-оригіналу (або цілого класу об'єктів), зокрема, досліджувати відгук фізичної системи, що моделюється на зміни її параметрів та початкових умов.

Побудова комп'ютерної моделі базується на абстрагуванні від конкретної природи явищ або досліджуваного об'єкта-оригіналу і складається з двох етапів – спочатку створення якісної, а потім і кількісної моделі. Само комп'ютерне моделювання полягає у проведенні серії обчислювальних експериментів на ПК, метою яких – є аналіз, інтерпретація та зіставлення результатів моделювання з реальною поведінкою об'єкта дослідження та, за необхідністю, подальше уточнення моделі.

Проектування об'єктів машинобудування, промислового й цивільного будівництва, радіоелектроніки й радіотехніки вступає в новий етап свого розвитку, коли разом зі зростанням складності проектів мають забезпечуватися скорочення термінів проектування й зменшення числа проектувальників значною мірою за рахунок автоматизації проектування й комп'ютеризації інженерної праці.

Розв'язання поставлених проблем неможливе без глибокого проникнення в фізичну сутність досліджуваних явищ, розробки та вдосконалення відповідних теоретичних положень, впровадження досягнутих результатів у виробництво. Геометричні методи давно та успішно використовуються в багатьох галузях промисловості. Велику роль тут мають відіграти нові методи геометричного моделювання та їх реалізація в системах комп'ютерної графіки, що дозволить розв'язувати задачі спеціальних дисциплін.

Нижче розглянуто основні геометричні методи, що використовуються при графічному моделюванні об'єктів, процесів, явищ в техніці та тенденції їхнього розвитку. За напрямок дослідження обирається твердотільне моделювання об'єктів, що утворюються (та змінюються в часі) під впливом різних зовнішніх чинників.

Триви́мірний о́пис об’є́кта (англ. 3D) – представлення об’єкта в трьох просторових вимірах. Як правило, ці виміри представлені в вигляді координат X, Y, та Z. Можливо мати дані з ідентичними координатами x та y при відмінній координаті Z. Наприклад, для цифрового представлення океанічних потоків, використовують 3D [15].

Твердотільне моделювання – це найдосконаліший і достовірний метод створення копії реального об'єкта, природний спосіб вираження сутності

вироби. Для створення таких моделей існують два методи уявлень: метод граничного (B-Rep) і метод конструктивного (C-Rep). В системі з C-Rep поданням моделі будуються з твердотільних примітивів. Ці примітиви визначаються розмірами, орієнтацією, формою і точкою прив'язки. Інструментами побудови C-Rep є булеві операції, вони базуються на алгебраїчної теорії множин. Найчастіше використовуються операції: різниця, перетин і об'єднання.

Ре́ндеринг (англ. rendering – відтворення, відрисовування) – в комп'ютерній графіці – це процес отримання зображення за моделлю за допомогою комп'ютерної програми [16]. Тут модель – це опис тривимірних об'єктів (3D) на строго певній мові або у вигляді структури даних. Такий опис може містити геометричні дані, положення точки спостерігача, інформацію про освітлення. Зображення – це цифрове растрове зображення. Зазвичай під рендерингом розуміють накладення текстури на уже готову твердотільну модель (solid-works) у механіці та на каркас (framework) в інженерній графіці.

Трасування променів (англ. ray tracing) у комп'ютерній графіці – є способом створення зображення тривимірних об'єктів чи сцени за допомогою відстеження ходу променя світла крізь точку екрану і симуляції взаємодії цього променя з уявними об'єктами, що підлягають відображенню [17]. Цей спосіб дозволяє створювати надзвичайно реалістичні зображення, зазвичай значно вищої якості, ніж дає типовий алгоритм Scanline або ж метод кидання променів (англ. Ray casting), проте має значно вищу обчислювальну складність. Із цієї причини алгоритми трасування променів використовуються там, де немає суттєвих обмежень часу рендерингу.

Граничне подання – це опис меж об'єкту або абсолютного аналітичного завдання граней, що описують тіло [18]. Метод B-Rep єдиний, дозволяє створити якісне зображення геометричного твердого тіла. Щоб встановити взаємну відповідність, потрібно задати кордону або контури об'єктів, а також ескізи різних видів об'єктів, і вказати лінії зв'язків між даними видами. У кожного з цих методів створення об'ємних моделей є плюси і мінуси, в порівнянні з іншим. У системи з C-Rep поданням перевага полягає в первинному формуванні моделі. Крім того, дане уявлення забезпечує більш зручне опис моделей в БД. B-Rep метод актуальний в утворенні складних структур, які дуже складно відтворити за допомогою C-Rep методу.

Перевагою систем з B-Rep є найпростіше зміна граничного подання в дану каркасну модель і зворотним її зміною. Причиною тому – є те, що опис меж аналогічно опису каркасною моделі. Проектування литтєвий оснащення та ливарних форм є традиційною областю суцільного, об'ємного

моделювання з імітацією руху. Самим очевидним відмінністю від двовимірного креслення є точне створення об'ємної комп'ютерної моделі.

Швидке формування креслень – одне з головних переваг тривимірного моделювання. Використовувати результати моделювання можна і на подальших стадіях розробки виробів – це є ще однією перевагою твердотільного моделювання [19].

На ранній стадії комп'ютерного проектування – комп'ютерна графічна модель була успішно використана як 2D програмне забезпечення для розробки та створення креслень на ПК. Комп'ютерна графічна модель складається з простих ліній, кіл, кривих та інших 2D-елемнтів. Через двовимірний характер, вона робить інтерактивне введення дуже легким. Користувач може просто використовувати мишу або інші пристрої введення, щоб ввести 2D-данні, а потім комп'ютер може генерувати необхідні графічні об'єкти.

1.6 Основоположні принципи відкритого програмного забезпечення

Відкрите програмне забезпечення (Open-source software) – програмне забезпечення, для якого є доступними (вихідний) програмний код, що забезпечує найкращі умови для вивчення такого програмного забезпечення та можливого подальшого внесення змін (удосконалень тощо) до нього.

Досить часто це поняття вважають тотожнім вільному ПЗ, що не є абсолютно правильним. Найістотніша відмінність полягає в тому, що ліцензії на вільне ПЗ обумовлюють, що усі подальші модифіковані версії такого ПЗ теж повинні розповсюджуватись як вільні (тому часто говорять, що ліцензія на вільне ПЗ є «інфікуючою»), в той час як більшість ліцензій на ПЗ з відкритими кодами надають повну свободу авторам модифікованих версій. В результаті вільне ПЗ завжди є ПЗ з відкритими вихідними кодами, але зворотне – є вірним далеко не завжди.

Наведемо декілька основних визначень та понять, що стосуються відкритого ПЗ та технологій проектування ГДБ.

Сирцевий код (зазвичай просто «сирці», також «вихідний код», «програмний код», «джерельний код», «текст програми», англ. «source code», рос. «исходный код») – будь-який набір інструкцій або оголошень, написаних комп'ютерною мовою програмування і у формі, що її може прочитати людина. Сирцевий код дозволяє програмісту спілкуватися з комп'ютером за допомогою обмеженого набору інструкцій.

Сирцевий код, що представляє собою програму, як правило, міститься в одному або більше текстових файлах, іноді зберігається у БД, як збережені

процедури, а також може з'явитися, як фрагменти коду, надруковані в книжках або інших засобах друку. Велика колекція файлів сирцевого коду може бути організована у дерево каталогів, і, в цьому випадку, воно може бути також відоме як дерево сирців (англ. source tree) або дерево програмного коду, дерево вихідного коду і т.п.

Сирцевий код програми – це набір файлів, потрібних для перетворення з форми, доступної для читання людині, на деякі види комп'ютерного виконуваного коду. Можливі два напрямки виконання сирцевого коду: транслюється у машинний код за допомогою компілятора, призначеного для певної комп'ютерної архітектури, або виконується безпосередньо за текстом за допомогою інтерпретатора.

Одна з перших САПР з такими характеристиками з’явилась тому, що у 1979 році балістична науково-дослідна лабораторія армії США (U.S. Army Ballistic Research Laboratory (BRL), зараз – United States Army Research Laboratory, висловила потребу в інструментах, які могли б допомогти з комп'ютерного моделювання та інженерного аналізу бойових систем корабля та умов роботи.

Коли жодна з САПР, які існували на той час виявились неадекватними для цієї мети, розробники ПЗ BRL почали збирати набір утиліт здатних інтерактивно переглядати і редагувати геометричні моделі. Цей пакет став відомий як BRL-CAD. Перший публічний реліз був зроблений у 1984 році. BRL-CAD став проектом з відкритим кодом в грудні 2004 року.

З того часу цей проект постійно розвивається, зявляються нові можливості. Дуже важливо що BRL-CAD розповсюджується на умовах ліцензій BSD та LGPL.

GNU – Lesser General Public License (загально-громадська ліцензія обмеженого використання) раніше GNU Library General Public License (загально-громадська ліцензія GNU для бібліотек) або LGPL – безкоштовна ліцензія на ПЗ, видана «Фондом Вільних Програм».

1.7 Огляд основних САПР-аналогів для роботи з ГБД

1.7.1 FreeCAD

У середовищі фахівців ряду галузей відома проблема створення повноцінної САПР в рамках open source, і хоча FreeCAD на момент створення цієї дипломної роботи, ще не є кандидатом на таку «повноцінність» – цей продукт може розглядатися як одна зі спроб створення бази для вирішення цієї проблеми.

Розробник FreeCAD Юрген Ригель, що працює в корпорації DaimlerChrysler, позиціонує свою програму як перший безкоштовний інструмент проектування механіки (порівнюючи свій продукт з такими розвинутими пропрієтарними системами як CATIA версій 4 і 5, SolidWorks), створений на основі бібліотеки Open CASCADE.

Мета створення цієї САПР – надати базовий інструментарій цієї бібліотеки в інтерактивному режимі.

1.7.2 QCad Community Edition

QCad – 2-мірна САПР з відкритим вихідним кодом, призначена для створення машинобудівних креслень та архітектурних планів. Працює під Windows, Mac OS X та на *nix системах. Має широке застосування у таких країнах, як Тайвань, іноді навіть як стандартне рішення компанії.

QCad надає різні інструменти для креслення, може працювати з растровими зображеннями і містить безліч інших інструментів. QCad не підтримує DWG-файли, які використовуються в AutoCAD. Однак файли DXF, з якими працює QCad, можуть бути відкриті практично у будь-якій CAD. Крім того, існують конвертери з DWG до DXF.

Випуск вільної версії QCad Community Edition відстає від професійної версії. Наприклад, остання версія Community Edition – 2.0.5.0, в той час як професійної – 2.2.2.0. Вільна версія поширюється у вигляді вихідних кодів за ліцензією GNU GPL. Доступна повнофункціональна демо-версія існує з обмеженнями за часом функціонування.

За своєю функціональністю програма дещо поступається пропрієтарним аналогам типу AutoCad, але є однією з небагатьох повноцінних 2D САПР під Linux / Unix, що має безкоштовну версію. Працює з використанням графічної бібліотеки Qt.

Існує форк QCad Community Edition, під назвою LibreCAD, створений для прискорення розвитку вільної версії силами незалежного спільноти. Зокрема в рамках проекту здійснюється перехід із застарілою бібліотеки Qt 3 на версію Qt 4, а також інтеграція засобів для експорту даних в систему EMC2.

1.7.3 SALOME

SALOME – відкрита інтегрована платформа для чисельного моделювання. Являє собою набір пре-і постпроцессинга. З початку була задумана як сполучне ПЗ CAD-CAE, що об'єднує в собі різні модулі, які

застосовуються в програмах чисельного моделювання – від моделювання у САПР до паралельних обчислень.

САПР-засоби у SALOME мають досить тісний зв'язок з платформою Open CASCADE Technology. Продукти марки SALOME поширюються на умовах GNU Lesser General Public License. Платформа SALOME використовується як база для проекту NURESIM (European Platform for NUclear REactor SIMulations), який призначений для повномасштабного моделювання ядерних реакторів.

В основі SALOME насамперед лежить концепція об'єктно-орієнтованого програмування. SALOME являє собою набір модулів, які дозволяють виконувати завдання, перелічені вище.

1.7.4 Electric VLSI Design System

Electric VLSI Design System – САПР, використовувана для розробки електричних схем і проектування топології друкованих плат та інтегральних схем (ІС). Крім іншого, це зручний інструмент для використання мов опису апаратури, таких як VHDL і Verilog.

Electric був open-source проектом протягом багатьох років, і зараз він легко доступний через FSF (Free Software Foundation).

Electric VLSI – САПР надвеликих інтегральних схем (НВІС). За допомогою Electric можна розробляти інтегральні МОП і біполярні схеми, друковані плати або схеми будь-якого типу. Electric має безліч стилів редагування, що включають планування, схематику, ілюстрації, архітектурне проектування. Electric може взаємодіяти з різними специфікаціями і форматами файлів як VHDL, CIF, GDS II.

Найбільш цінна вбудована в Electric можливість – це система прив'язок, яка дає можливість здійснювати проектування зверху вниз з дотриманням цілісності всіх з'єднань. Екран показує реальну геометричну форму, але це означає і зв'язаність теж. САПР має звичайний інтерфейс користувача. Одна САПР, з єдиним інтерфейсом, може бути використана для створення як топології, так і електричних схем. Electric щільно інтегрує процес малювання, відокремлюючи схематику, і має LVS інструмент для їх порівняння.

Недоліки топологічного проектування, заснованого на пов'язаності також відомі. Воно відрізняється від всіх інших і вимагає перепідготовки. Це дійсно так, але багато перевчилися і знайшли його вартим. Користувачі, яким добре знайома геометрична компоновка топології ІС зазвичай навчаються довше і важче. Electric підходить для тих хто не має досвіду проектування ІС.

1.7.5 KiCad

KiCad – розповсюджуваний за ліцензією GNU GPL програмний комплекс класу EDA з відкритими вихідними текстами, призначений для розробки електричних схем і друкованих плат.

Кросплатформеність компонентів KiCad забезпечується використанням бібліотеки wxWidgets. Підтримуються операційні системи Linux, Windows NT 5.x, FreeBSD і Solaris.

Розробник – Жан-П'єр Шарру (фр. Jean-Pierre Charras), дослідник в LIS (фр. Laboratoire des Images et des Signaux – лабораторія зображень і сигналів) та викладач електроніки та обробки зображень у IUT de Saint Martin d'Hères (Франція).

Програмні модулі, що входять до KiCad:

– KiCAD – менеджер проектів;

– eeschema – редактор електричних схем;

– вбудований редактор символів схем (бібліотечних компонентів);

– pcbnew – редактор друкованих плат;

– вбудований редактор образів посадочних місць (бібліотечних компонентів);

– 3D Viewer – 3D-переглядач друкованих плат на базі OpenGL (частина pcbnew);

– gerbview – переглядач файлів Gerber (фотошаблонів);

– cvpcb – програма для вибору посадкових місць, відповідних компонентів на схемі;

– wyoeditor – текстовий редактор для перегляду звітів.

Компоненти KiCad eeschema забезпечують функції:

– створення однолистових та ієрархічних схем;

– перевірку їх коректності ERC (контроль електричних правил);

– створення списку електричних ланцюгів netlist для редактора топології плати pcbnew або для Spice-моделювання схеми;

– доступ до документації на схеми, що використовуються, та електронні компоненти (datasheet);

– розробку плат, що містять до 16 шарів міді і до 12 технічних шарів (шовкографія, паяльна маска та ін.);

– вихід на зовнішні трасувальні з'єднанняь за допомогою генерації опису плати на Specctra Design Language (on-line FreeRoute тощо);

– генерацію технологічних файлів для виготовлення друкованих плат (Gerber-файли для фотоплоттерів, файли сверловок та файли розміщення компонентів);

– пошарова друк схем і креслень друкованих плат на принтері або плоттері (у форматах PostScript, HPGL, SVG і DXF), з рамкою формату або без неї;

– gerbview дозволяє переглядати Gerber-файли.

1.7.6 Wings 3D

Wings 3D – це безкоштовна програма 3D-моделювання з відкритим вихідним кодом, на яку вплинули програми Nendo і Mirai від компанії Izware. Програма отримала назву за назвою технології обробки полігонів. Більшість користувачів називають її просто Wings.

Wings 3D доступна для багатьох платформ, включаючи Windows, Linux і Mac OS X. Програма використовує оточення і мову програмування Erlang. Wings 3D використовує контекстне меню, в протилежність насиченому графічному інтерфейсу. Перемикання різних методів редагування (вертекс, ребра, грані і об'єкти) здійснюється мишкою і клавіатурою.

Тому, що Wings спроектований для використання контекстного меню, кожен з методів має свій власний набір команд для редагування моделі. Багато з цих команд одночасно мають прості і складні налаштування, що відносяться до того, як саме цей інструмент буде впливати на редаговану модель. ПЗ також може накладати текстури і матеріали на модель і має вбудований автогенератор текстурних координат.

Особливості:

– різноманітні інструменти виділення і моделювання;

– інструмент моделювання підтримує примагнічення та векторні операції;

– настроювання інтерфейсу користувача та гарячих клавіш;

– режим Tweak Mode, що дозволяє швидко змінювати модель;

– підтримка джерел освітлення, матеріалів, текстур, вершин;

– авторозгортання;

– присутність менеджеру розширень;

– можливість імпорту та експорту потоків сформованих даних у найпопулярніші графічні формати.

2 ПРОГРАМНА ЧАСТИНА

2.1 Загальні відомості про створення програми BRL

З кінця 1950-х комп'ютери використовувалися для розрахунків, пов'язаних з розробкою систем озброєнь. Цей важкий труд вимагав величезної кількості грошей і часу – як для того, щоб креслення перетворилися на повномасштабну модель, так і для вдосконалення при тестуванні і проведенні випробувань.

У 1979 р. Військова балістична лабораторія США (US Army Ballistic Research Laboratory (BRL), нині відома як US Army Research Laboratory (ARL) – Військова дослідна лабораторія) відчувала гостру потребу в коштах комп'ютерного моделювання систем озброєння – танків, ракет, літаків та ін.

Оскільки жодна з існуючих на той момент САПР не була визнана придатною, програмісти BRL приступили до розробки власного пакету додатків, призначеного для відображення, редагування та суміщення геометричних моделей. Так був створений BRL-CAD – пакет додатків для твердотільного моделювання (constructive solid geometry, CSG). Завдяки приблизно мільйону рядків С-коду BRL-CAD став найпотужнішим пакетом графічного моделювання, ліцензованим більш в чому 2 тис. організацій по всьому світу. Він містить велику кількість інструментів, утиліт і бібліотек, таких як:

а) MGED (Multiple-Device Geometry Editor) – незалежний від пристрою виводу інтерактивний редактор геометрії;

б) RT (ray tracer) – засіб візуалізації на основі самого правильного методу; трасування променів світла);

в) nirt (інтерактивний трасувальник променів);

г) remrt (мережевий трасувальник для розподілу складних завдань візуалізації).

BRL-CAD підтримує одночасно ще два способи взаємодії з користувачем – за допомогою командного рядка і графічного інтерфейсу користувача (GUI).

BRL-CAD підтримує велику різноманітність геометричних уявлень включаючи великий набір традиційних CSG примітивних неявних твердих речовин, таких як коробки, еліпсоїди, конуси і тори, а також явні тверді із закритих колекцій уніформи, B-сплайнових поверхонь, нерівномірні Rational B-Spline (NURBS) поверхонь, n-різноманітну геометрію (НРГ), суто грановані сітки геометрії. Всі геометричні об'єкти можуть бути об'єднані, використовуючи логічні теоретико-множинні операції, включаючи CSG об'єднання, перетину і різниці.

Фундаментальна властивість пакету полягає в його здатності конструювати та аналізувати реалістичні моделі на основі складних об'єктів, що складаються з відносно невеликого набору графічних примітивів (primitive shapes). Для побудови складних об'єктів використовуються булеві операції: об'єднання, віднімання і перетин. Ще одна сильна сторона пакета – швидкість засобів візуалізації, трасувальника світла, який є одним з найшвидших серед існуючих. І нарешті, користувачі BRL-CAD можуть вимальовувати моделі з великою точністю, від субатомних до галактичних масштабів за принципом «все деталі, весь час».

Американські військові після 20 років використання BRL-CAD для своїх потреб раптом відкрили вихідний код програми. Може бути, що взяли на озброєння щось більш потужне. Але факт залишається фактом: система автоматизованого проектування BRL-CAD, що використалася ще Військової дослідною лабораторією США на системах PDP-11, поширюється тепер безкоштовно, з відкритими вихідниками.

Спочатку були відкриті і безкоштовні версії для SGI Irix (версія Unix від фірми Silicon Graphics), потім для Linux і нарешті в 2005 р. з'явилася версія для платформи Win32.

BRL-CAD реалізує підхід Unix. Це середу моделювання, що складається з безлічі невеликих модулів. Можна використовувати будь-який компонент або розробити свій. Багато хто критикує інтерфейс редактора MGED з поставки пакета. Тому на сьогоднішній день розроблено декілька альтернативних редакторів. У Windows-версію включений один з них. Це більш дружній, ніж MGED, і такий же гнучкий інструмент Archer (hbc/3).

У порівнянні з MGED Archer більше орієнтований на використання миші для перегляду і редагування моделі. За допомогою спеціальних кнопок на основній панелі інструментів можна створювати графічні примітиви. Будь створені об'єкти, в тому числі і комплексні, відображаються в дереві об'єктів в лівій частині вікна.

Обравши один з них – можна поміняти його властивості на панелі праворуч. І нарешті, у правого краю вікна розташована панель з кнопками, що дозволяють перемикатися між режимами обертання, огляду, масштабування і центрування всієї моделі або вибраного об'єкта. Працює на платформах: BSD, IRIX, Linux, MacOS X, Solaris и Windows.

Особливості програми BRL-CAD:

- BRL-CAD включає в себе декілька функцій, які роблять його використання корисним для розробки ігор;
- високопродуктивне точне виявлення зіткнень за допомогою реального часу трасування променів;

- підтримка розширеного геометричного уявлення, у тому числі РГС і сітки об'єктів;
- компактна геометрія, двійковий формат файлу для зберігання ігрових об'єктів;
- процедурна API геометрія для створення геометрії за допомогою коду;
- інструменти для обробки сигналів та зображень;
- інструменти для перетворення геометрії та маніпуляції;
- використання бібліотеки транспортної мережі;
- документально висувні інструменти моделювання інтерактивного редактору в світі 3D.

Крім цього, дана система широко використовується в різних військових, академічних та промислових додатках, включаючи системи проектування та аналізу автомобілів, механічних вузлів і архітектурних споруд. Також система використовувалася для визначення ступеня радіоактивного зараження, медичної візуалізації, навчанні комп'ютерної графіки, принципам конструювання та моделювання суцільних тіл, вимірі продуктивності обчислювальних пристроїв та ін.

2.2 Інструменти та утиліти BRL-CAD

BRL-CAD це колекція з більш ніж 400, утиліт та додатків, що містить понад мільйон рядків вихідного коду. Пакет спеціально розроблений, в значній мірі як крос-платформна система, що активно розвивається і підтримується на багатьох поширених операційних середовищах, включаючи для BSD, Linux, Solaris, Mac OS X і Windows, та інші. BRL-CAD поставляється тільки в бінарній формі і вихідного коду, як безкоштовне програмне забезпечення з відкритим вихідним кодом (FOSS), що надаються на умовах відкритого вихідного коду (OSI), затверджених умов ліцензії.

Додаток з боку BRL-CAD також пропонує ряд інструментів і утиліт. Вони в першу чергу стосуються геометричних перетворень, зображення формату перетворення, і командного рядка-орієнтоване зображення маніпуляції. Нижче наведено список основних BRL-CAD інструментів і утиліт:

а) MGED – графічний редактор BRL-CAD;

б) інструменти для трасування променів і допит трасування променів геометричних об'єктів;

в) RT – головний Raytracer для рендеринга зображень в BRL-CAD;

г) Nirt – пакет для стрільби променів в інтерактивному режимі та отримання інформації про те, що промені зіткнулися;

д) remrt – мережа розподіленого трасування пакетів.

Асортимент геометричних конвертерів для перетворення в інші геометричні формати (людський фактор модель для виконання навантаження)

Реальна мова розмітки (VRML), стереолітографії (STL), Cyberware Digitizer даних, а FASTGEN4.

bwish – TCL / TK перекладача у віконній оболонці з аксесуарами компілірованний для доступу BRL-CAD бібліотеки.

irprep – виробляє вхід PRISM (фізично реалістична інфрачервона імітаційна модель).

JOVE (власна версія Джонатана Emacs) – це швидка, легка реалізація Emacs.

Програми для відображення зображень різних типів на фрейм буфері застосування і отримання даних з буфера кадрів.

Інструменти для створення геометрії для спільних об'єктів, таких як паркани, стіни,математичні і геометричні дивацтва

Утиліти для створення анімації, скрипти – відстеження стовпчастих даних і інтерполяції це дозволяє зробити декілька елементів для анімації.

Юта растрових ToolKit – для роботи із зображеннями всіх RLE на основі зображень. Програми для роботи із зображеннями й перетворення між різними типами файлів зображень. Два основних BRL-CAD типів пікселів (24-біт червоний, зелений і синій [RGB] кольорового зображення) і маси тіла (8-бітові відтінки сірого зображення)

Програми для фільтрації зображень, робить гістограми на дані зображення, і витяг прямокутники із зображень

Інструменти для об'єднання двох зображень і змішування їх разом. Ці інструменти були створені для редагування зображень для відео BRL-CAD бібліотеки призначені в першу чергу для геометричного моделювання, який також хоче возитися з програмним забезпеченням і, можливо, створювати власні інструменти.

Функції: libbu – основні утиліти бібліотек, що містить фундаментальні типи процедур, в тому числі маніпулювання даними

2.3 Графічні бібліотеки BRL-CAD

BRL-CAD містить ряд інструментів для аналітичного трасування променів, в тому числі NIRT, rtarea, rtweight, rtcheck і g_qa. Ці окремі

інструменти можуть часто містити функціональні можливості та інші інструменти, що можуть використовувати методи, доступні тільки в одному інструменті. Отже, перелік бібліотек:

- libbn – це бібліотека підпрограм для підтримки основних частин, обробка, в тому числі 2-D/3-D вектора, матриці і кватерних маніпуляцій, а також вейвлет-декомпозиції та реконструкції та ін.;
- libdm - первинні графіки BRL-CAD-дисплей менеджера бібліотеки. Це ручки відкривання вікон на дисплеї і відображення геометрії у графічному вікні, і т.д.;
- libfb - кадровий буфер (КБ) бібліотекою, яка підтримує мають вікна в які користувач ставить піксельні дані;
- libmultispectral і liboptical - шейдеров і текстур бібліотеки для Raytracer;
- libpkg - бібліотека, яка реалізує віддалений виклик процедур (RPC) механізм. Ця бібліотека є попередником сучасної системою RPC. На відміну від типової служби RPC UNIX, можна налаштовувати послуги в додатках та обробляти запити, не вимагаючи конфігурації систем на адміністратор;
- librt - бібліотека, яка містить всю геометрію підтримки, в тому числі дані уявлення для примітивів, підтримка трасування променів (RT), і дискретних входів / виходів підтримки РГС геометричного опису;
- libz - суспільним надбанням Бібліотека стискання;
- libtcl, libtk, і libitcl - бібліотеки, які надають Tcl / Tk сценарії.

2.2.1 Використані команди BRL та їх можливості

Об’єкти примітивів можуть бути створені в модулі MGED. Створення об’єктів може відбуватися в один з способів:

- make. В MGED команда make створює об'єкт з розмірами за замовчуванням;
- in – інтерактивно запитує розміри вже не в якості аргументів;
- form – графічний редактор примітивних форм (деякі об'єкти не повністю підтримується);
- create – меню графічного створення об’єктів.

Модель в BRL-CAD є базою даних з розширенням *.g.

2.2.2 Домовленість про імена файлів

У BRL-CAD можливі наступні формати розширення файлів чи об’єктів, що використовуються:

а) двійкові файли баз даних BRL-CAD – *.g;
б) ascii-файли баз даних BRL-CAD (застаріли) – *.asc;
в) необроблені бінарники кольорових (RGB) картинок – *.pix;
г) необроблені бінарними ч/б картинок – *.bw;
д) розширений UNIX формат 2-х / 3-х мірних даних – *.pl;
е) команди для рендерингу збережених у файл сценаріїв – *.rt.

2.2.3 Домовленість про імена геометрії:

У BRL-CAD можливі наступні домовленості про імена геометрії:
а) групи або збірки з невизначеним суфіксом – *.g;
б) області / деталі – * .r;
в) комбінації тіл, що не є областями – *.c;
г) примітивні твердо тільні форми – *.s.

2.4 Створення простих тіл

Розпочнемо створення тіл та об’єктів у BRL-CAD зі створення моделі / бази даних. Для цього необхідно здійснити створення нової бази даних, для чого необхідно скористатися командою File / Open. Після чого в діалоговому вікні, яке відкрилося, прописати ім’я нової бази даних / моделі. Також після створення бази даних користувач може обрати / змінити одиниці вимірювання:

```
mged> units mm
```

В випадку зміни одиниць вимірювання, необхідно ввести units mm, щоб змінити на міліметри.

Код моделі:

```
mged> in base1.s arb8
Enter X, Y, Z for point 1: 0 0 0
Enter X, Y, Z for point 2: 20 0 0
Enter X, Y, Z for point 3: 20 10 0
Enter X, Y, Z for point 4: 0 10 0
Enter X, Y, Z for point 5: 0 0 5
Enter X, Y, Z for point 6: 20 0 5
Enter X, Y, Z for point 7: 20 10 5 0 10 5
```

```
base1.s
mged> in
Enter name of solid: w0.s
Enter solid type: rcc
Enter X, Y, Z of vertex: 0 0 0
Enter X, Y, Z of height (H) vector: -3 0 0
Enter radius: 2.5
w0.s
Unable to do <Edit Reject> from VIEWING state.
```

Запустимо написаний код в програмі BRL-CAD, а саме у модулі MGED (рисунок 2.1).

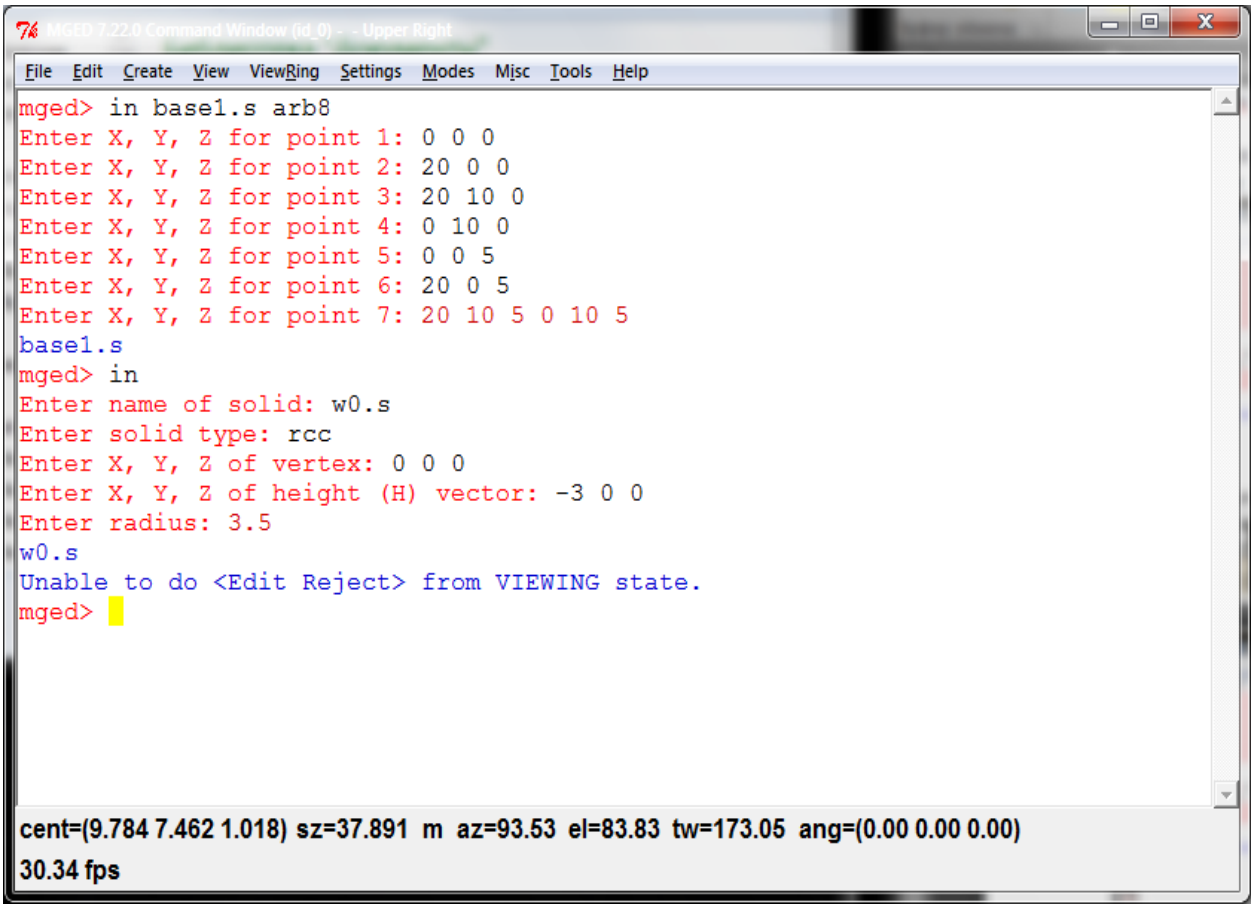

Рисунок 2.1 – Перегляд коду моделі в модулі MGED

В наведеному коді була використана команда in, яка створює нову примітивну форму за вказаними параметрами. Ім’я створеної форми – w0.s Тип примітиву, який створився після запуску коду моделі – rcc, тобто циліндр (right circular cylinder). Після введення команди in програма починає опитувати користувача для введення координат точок побудови майбутньої

моделі. Користувачеві необхідно вказати по три координати для кожної точки. Після закінчення вводу координат, програма відправляє запит на вказування:

а) ім’я форми;

б) координат вершини;

в) координат точок висоти вектору;

г) радіусу примітиву кола об’єкту.

Результат виконання коду моделі подано на рис. 2.2:

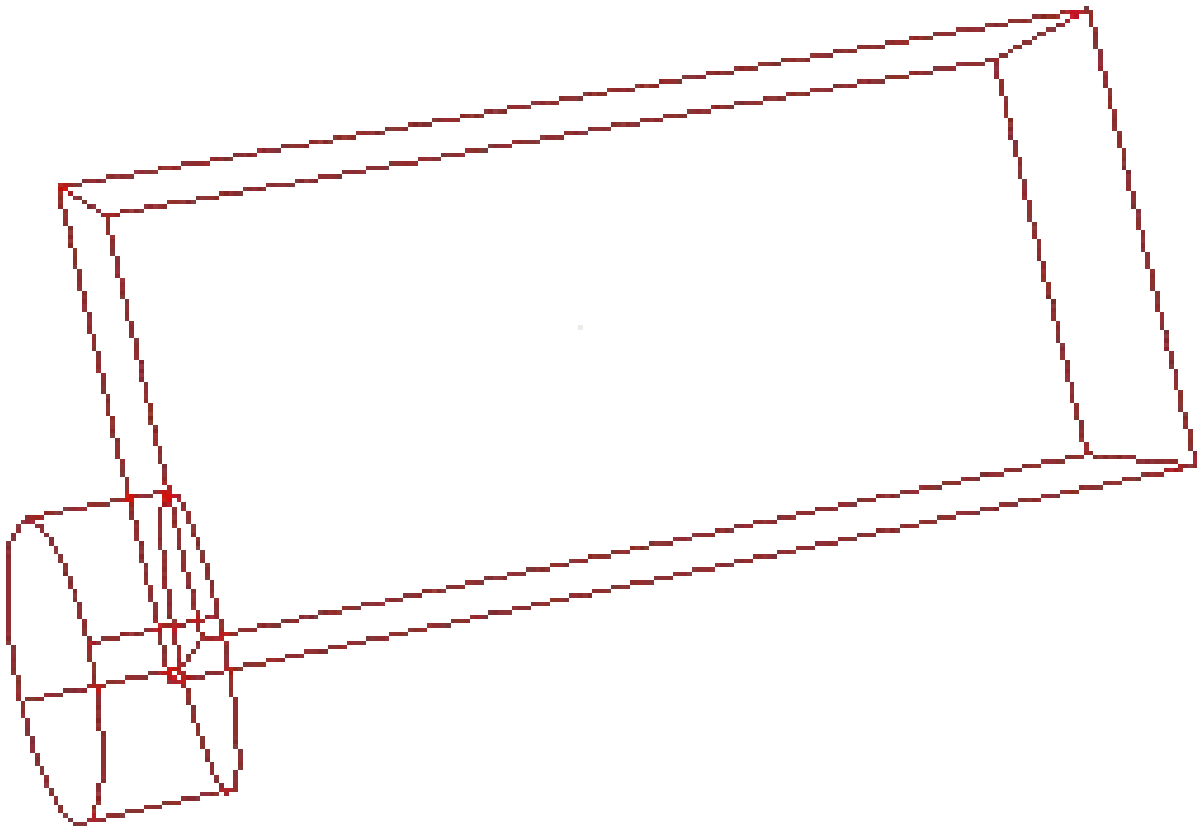

Рисунок 2.2 – Створена примітивна форма

Для роботи, тобто редагування, створеного об’єкту користувачеві необхідно знати про основні функціоналі клавіші програми BRL-CAD. Виділення тексту в командному вікні (КВ) можна зробити за допомогою затиснутої правої клавіші миші, копіювання і вставка відбувається за допомогою середньої клавіші миші. Натиснувши клавішу «M» в графічному вікні (ГВ) користувач побачить центр координат і напрям осей. В програмі передбачене авто доповнення (Tab як у консолі).

Тепер повернемо створений циліндр навколо осі Z на 90 градусів.

Для виконання повороту циліндра пропишемо код, який буде запущений в модулі MGED:

```
mged>sed w0.s
mged>arot 0 0 1 90
mged>accept
```

Пропишемо код в консольному вікні модуля MGED (рис. 2.3).

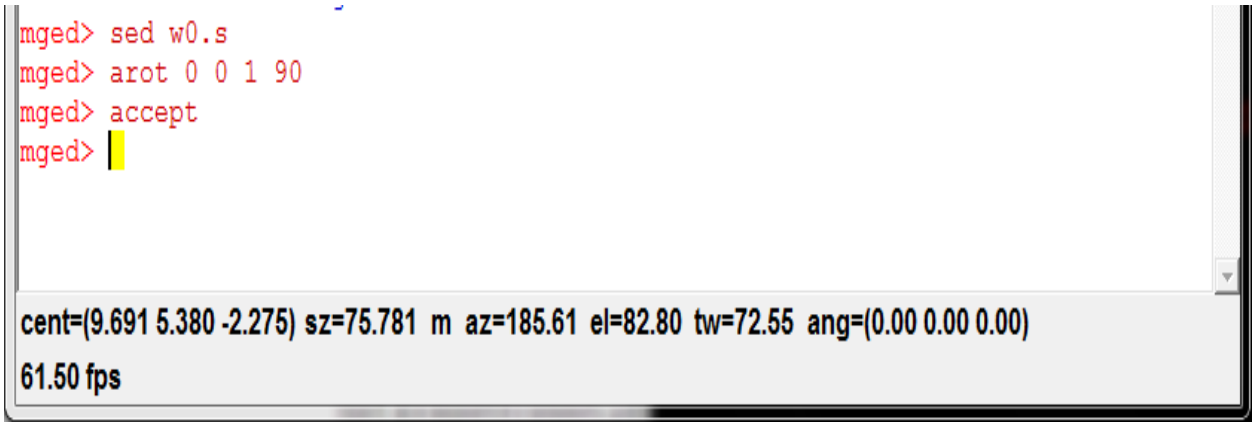

Рисунок 2.3 – Прописаний код для повороту циліндра

Команда sed переводить в режим редагування об'єкт, який вказаний після неї, в даному випадку вказана назва форми w0.s, тобто циліндр. Команда arot повертає на зазначений кут об'єкт навколо зазначеної осі.

В випадку, якщо користувач згоден зі внесеними змінами і хоче знову перейти до режиму перегляду створеного примітиву об’єкту необхідно ввести команду accept. А в випадку не згоди ввести до консольного вікна команду reject і все повернеться до моменту редагування.

Результат запуску коду приведено на рис. 2.4.

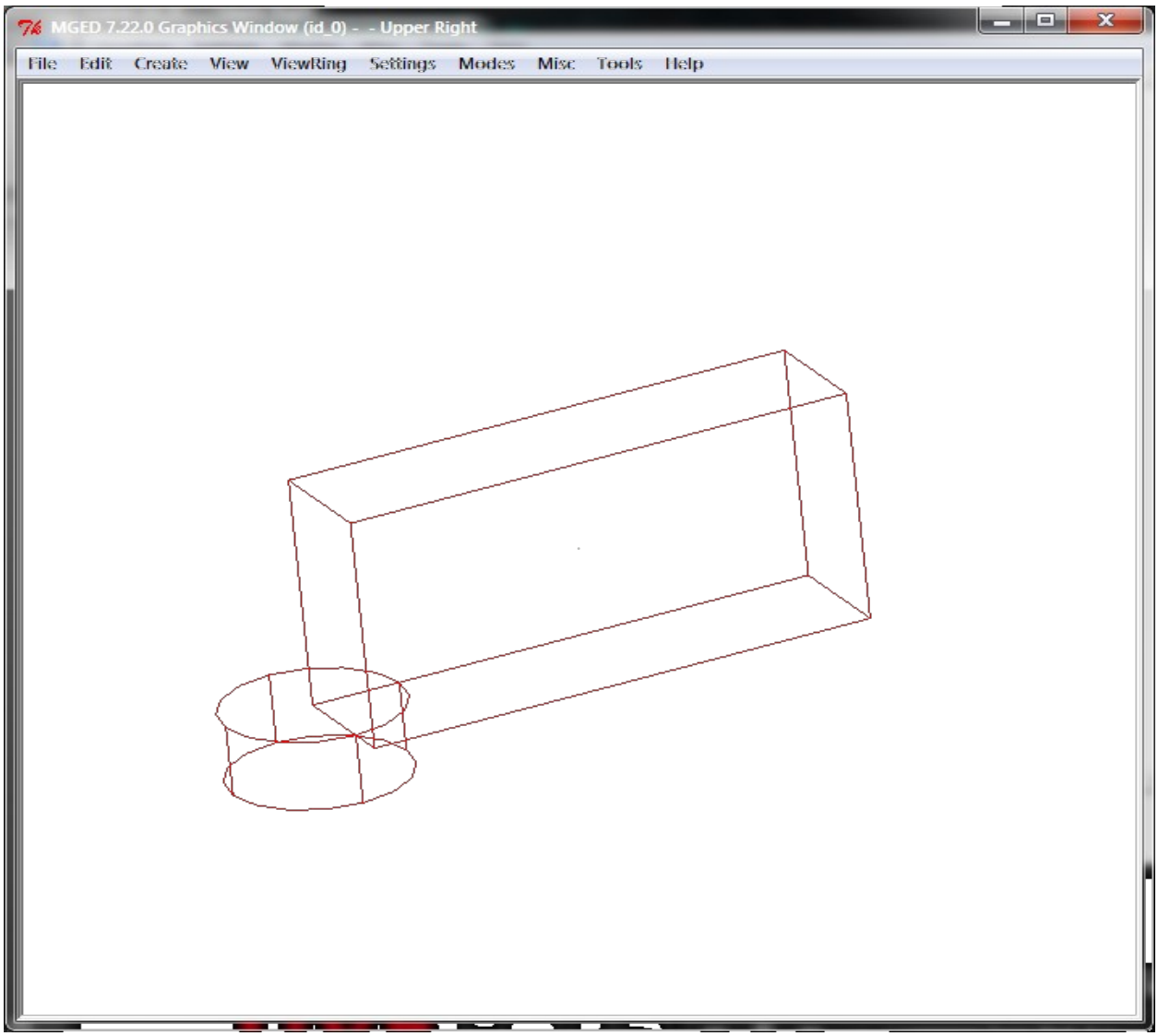

Рисунок 2.3 – Прописаний код для повороту циліндра

Наступним кроком виконання завдання – є прописання коду для створення тору:

```
mged> in len0.s tor
Enter X, Y, Z of vertex: 0 0 0
Enter X, Y, Z of normal vector: 0 -1 0
Enter radius 1: 3
Enter radius 2: 5
Error: ERROR, radius 2 >= radius 1 ....
in: ERROR tor not made!

mged> in len0.s tor 0 0 0 0 -1 0 5 1
len0.s
```

```
mged> kill len0.s
base1.s w0.s myPrimitive len0.s
mged> in len0.s tor 0 0 0 0 -1 0 5 3
len0.s
mged>
```

Пропис коду для створення тора в модулі MGED (рис. 2.4).

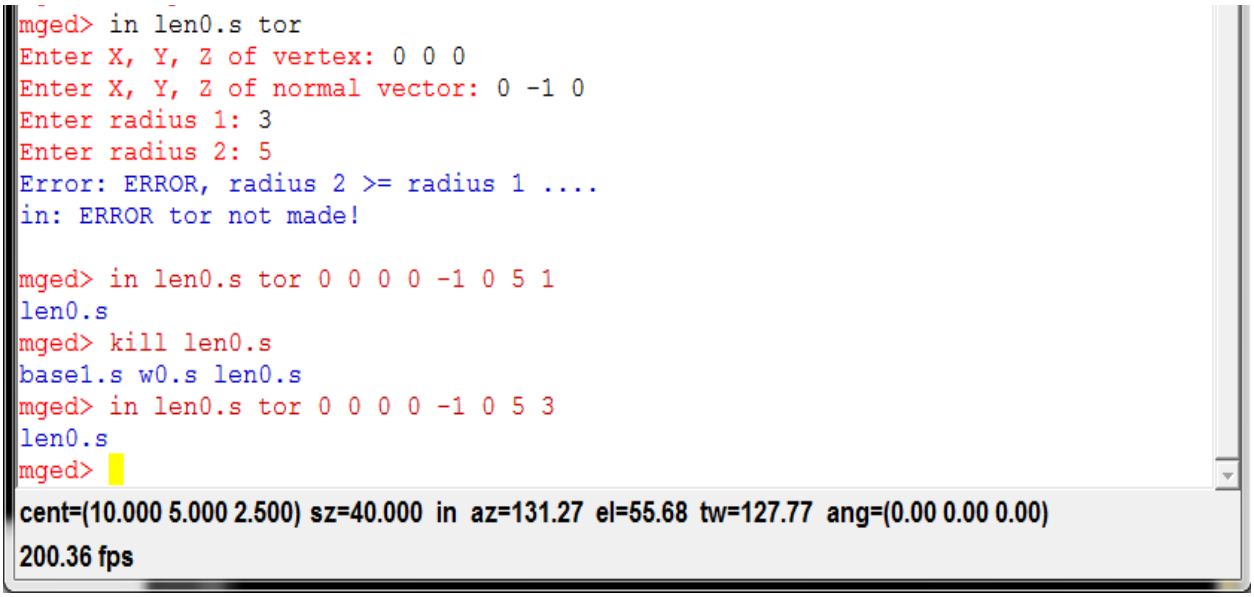

Рисунок 2.4 – Створений код моделі для тору

В наведеному коді, рисунок 2.4, при створенні тору користувач дізнається, що радіус 1 – це відстань від осі до центру кола в перетині тора, а радіус 2 – окружність, що обертається навколо осі тора. А використання команди kill забезпечує видалення непотрібних користувачеві об’єктів.

Вкладка Edit графічного вікна (ГВ) містить підпункт меню Browse Geometry, в якому відображаються існуючі геометричні форми (рис. 2.5). Клавіші стрілки в графічному вікні відповідають за повороти навколо Z і X.

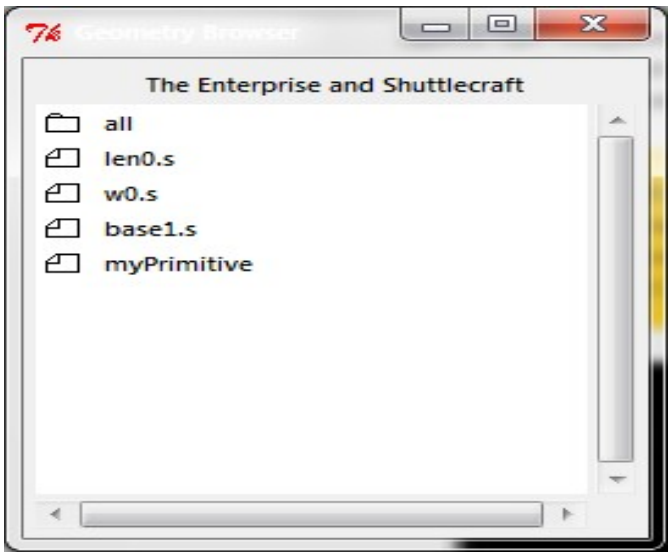

Рисунок 2.5 – Відображення створених об’єктів

Результатом запуску коду в модулі MGED САПР BRL-CAD – є створений тор (рис. 2.6).

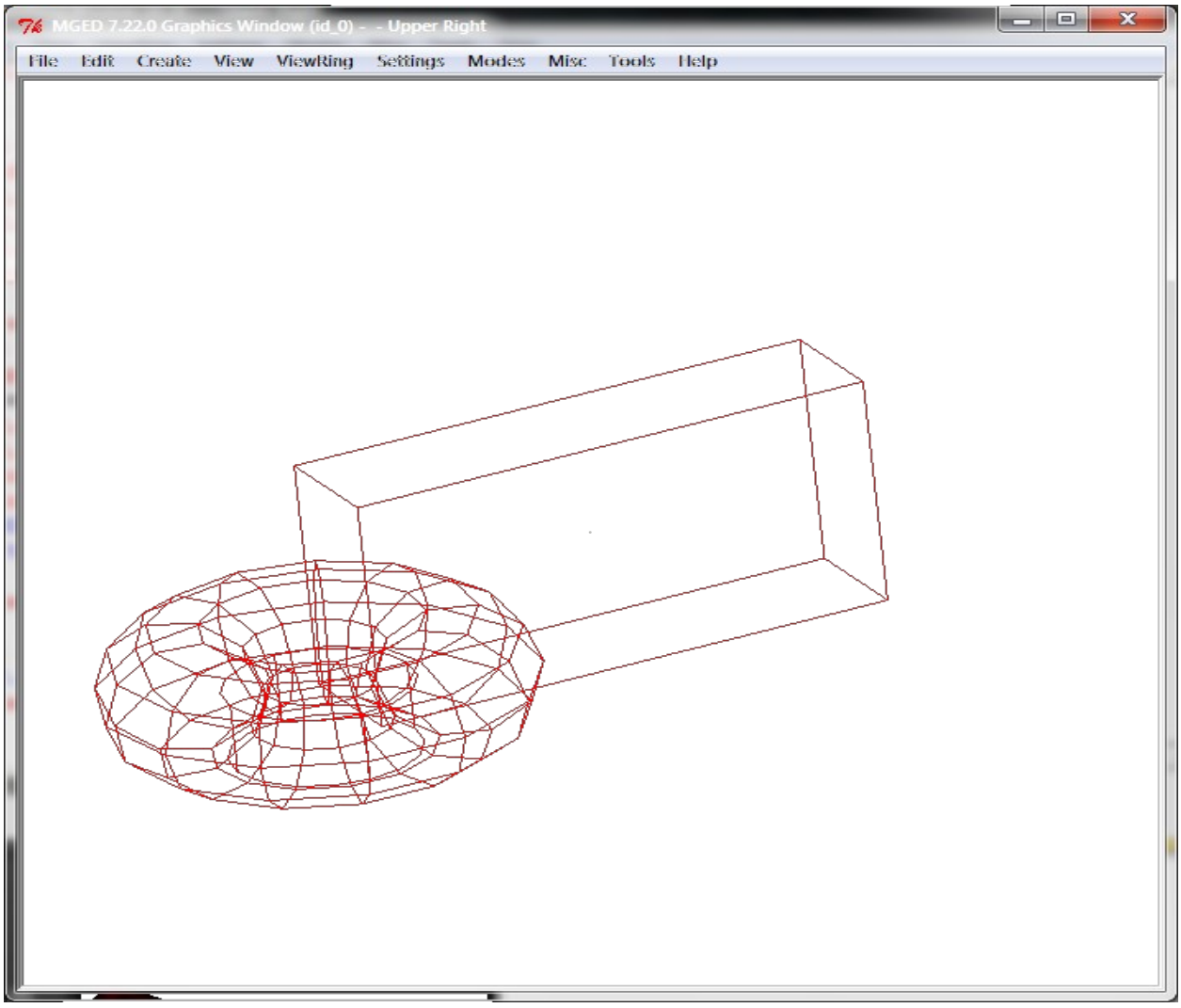

Рисунок 2.6 – Створений тор в програмі BRL-CAD

2.5 Логічні операції

Системою BRL-CAD підтримуються три типи логічних операцій, а саме: union (об'єднання), subtraction (віднімання) і intersection (перетин). Ці операції означаються операторами «u», «-» та «+» відповідно (MGED використовує їх в якості префіксів позначень). У цих позначеннях оператор об'єднання має найвищий пріоритет у порівнянні з іншими.

Так, що наступне логічне вираження – (A union B) subtract C можна записати, як:

u A - C u B – C.

Почнемо роботу з використанням логічних операцій. Написаний код спробуємо.

```
mged>comb wheel1.c u len0.s u w0.s
mged> Z
mged> e len0.s
mged> e w0.s
mged> Z
mged> e wheel1.c
mged>kill w0.s
wheel1.c
```

Результат виконання коду в програмі BRL-CAD показаний на рис. 2.7.

Тобто, використана команда comb утворює роботу з використанням логічних комбінацій. В першому рядку користувач об’єднує форму wheel1.c з об’ктом len0.s (тор).

Після вказаного об’єднання відбувається приєднання до створеного об’єкту деталі w0.s (циліндр). За допомогою команди Z видаляються всі об'єкти з графічного вікна (ГВ). Команда «e» показує зазначені об'єкти, так як насправді моделі або поверхні, які візуально виглядають складними на самому ділі є більш розумно скомбінованими або декомбінованими простими об'єктами.

Тобто, видаливши об'єкт попередник з бази даних, ми втратили інформацію не лише про нього, а й про тіла, в які він входив. Тому об'єкти попередники прибираються на час виконання роботи, а не видаляються зовсім.

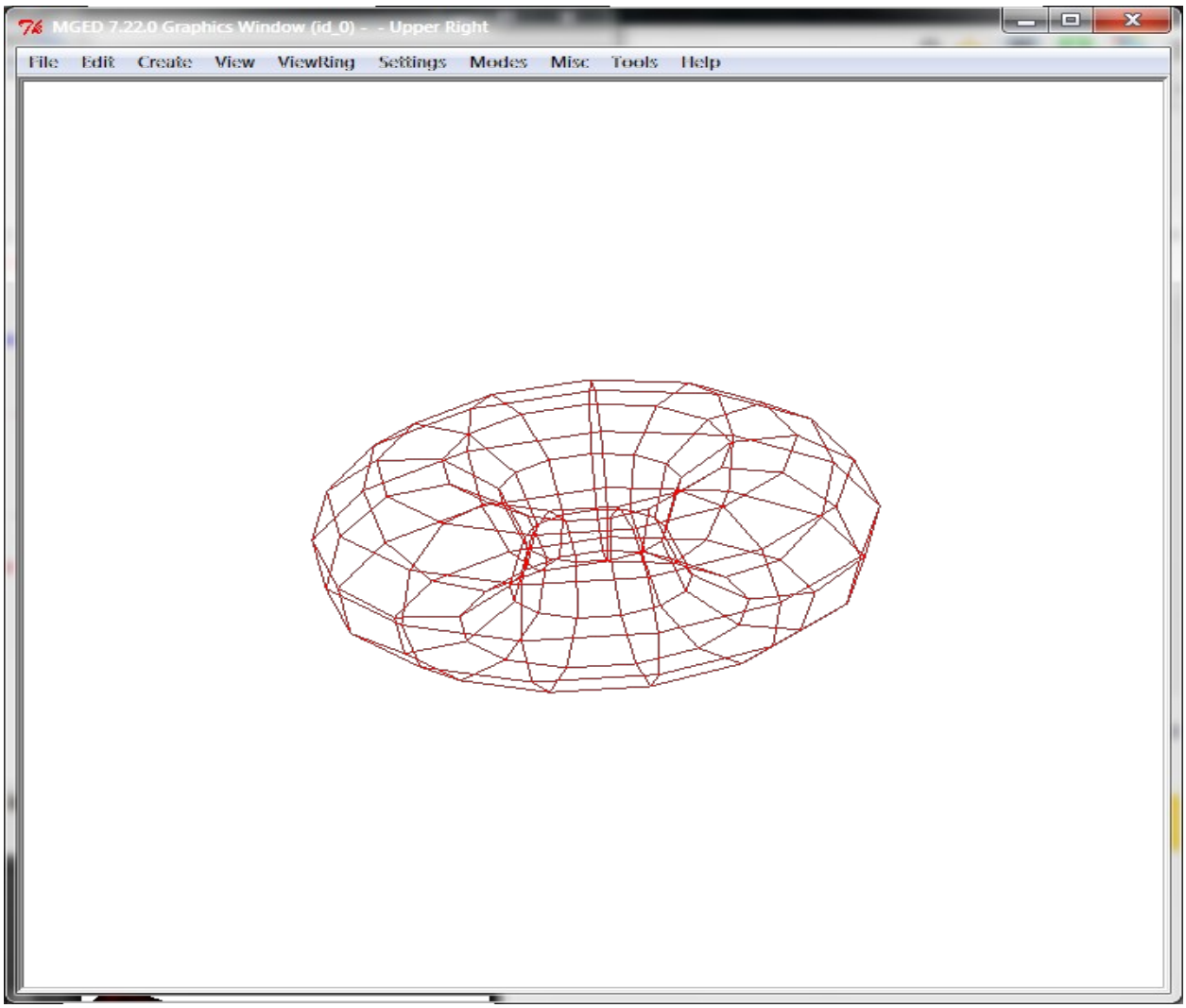

Рисунок 2.7 – Результат виконання логічної комбінації видалення форми w0.s (циліндр) з графічного вікна програми BRL-CAD

Після видалення з графічного вікна в результаті застосування логічних комбінацій циліндру, спробуємо прибрати з вікна форму wheel1.c, що в даному випадку забезпечить повне очищення графічного вікна від всіх деталей, що містилися в ньому. Реалізувати дане завдання можна за допомогою команди kill з зазначенням об’єкту, який необхідно прибрати з вікна реалізації, а саме деталі wheel1.c. Код для даної операції виглядає наступним чином:

```
mged>kill wheel1.c
wheel1.c
```

Переглянути результат застосування наведеного фрагменту програмного коду можна на рисунку 2.8, на якому наглядно показаний результат роботи програми з застосуванням команди kill.

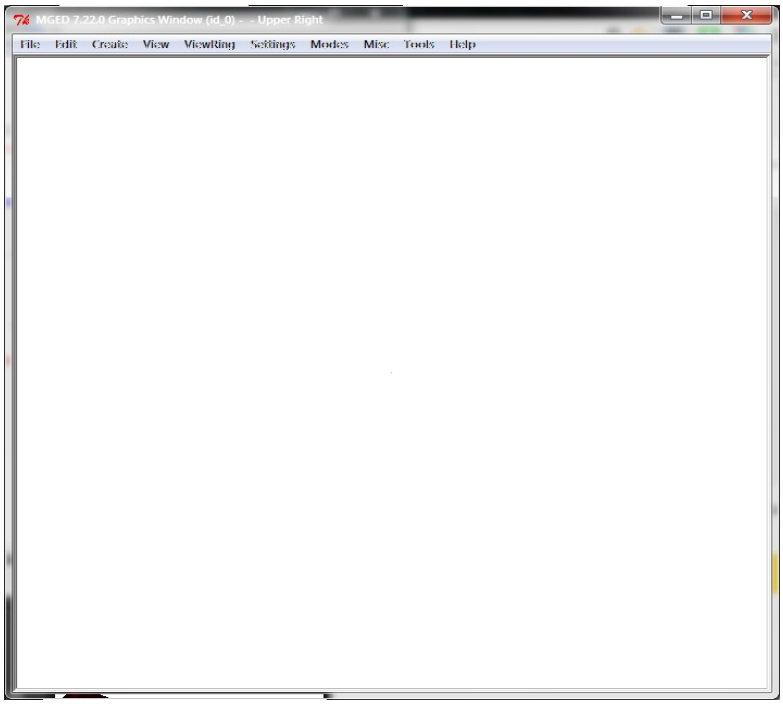

Рисунок 2.8 – Результат роботи програми після застосування команди kill

Після видалення всіх об’ктів, що містилися в графічному вікні, необхідно провести роботу для повернення видалених деталей. Для цього напишемо програмний код, який забезпечить відображення необхідних об’єктів на графічному вікні програми BRL-CAD.

```
wheel1.c
mged> in w0.s rcc 0 0 0 -3 0 0 2.5
w0.s
mged>sed w0.s
mged>arot 0 0 1 90
mged> accept
mged>ls
mged> e base1.s
mged> e len0.s
```

Задіяна в програмному коді команда sed дозволяє редагувати примітиви (в режимі твердого редагування). За допомогою команди arot користувач вказує градус кута повороту навколо довільної осі. Зазначена

команда accept дозволяє повернутися в режим перегляду, а також приймати будь-які правки. Використання команди ls дозволяє перерахувати всі об'єкти, що містяться у відкритій або створеній базі даних користувача.

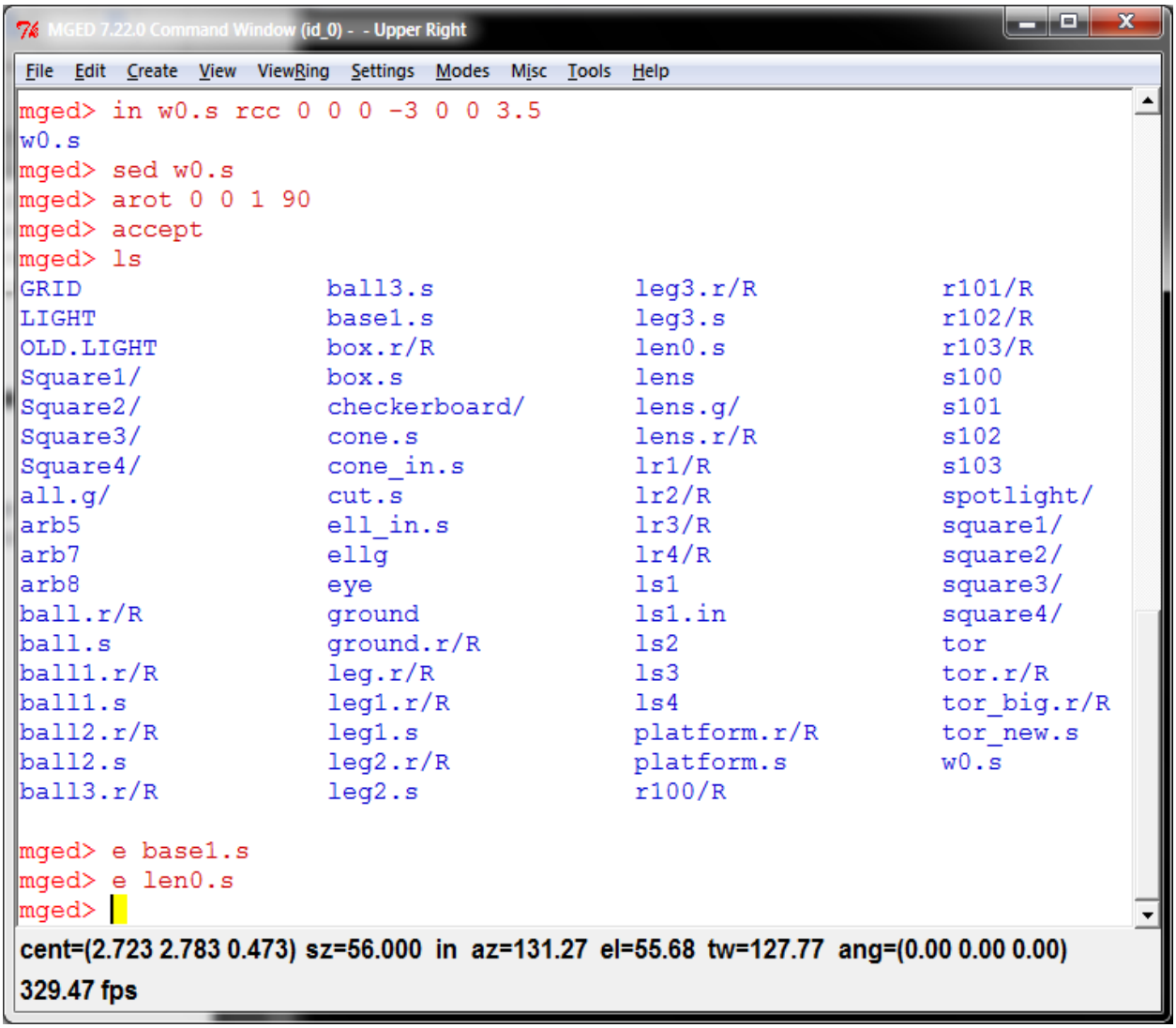

Рисунок 2.9 – Фрагмент програмного коду в консолі програми BRL-CAD

Для перегляду результату виконання коду перейдемо до графічного вікна програми BRL-CAD, що зображений на рисунку 2.10. На даному рисунку показаний результат роботи програмного коду, завдяки якому відбулося повне відображення об’єктів, які були створені.

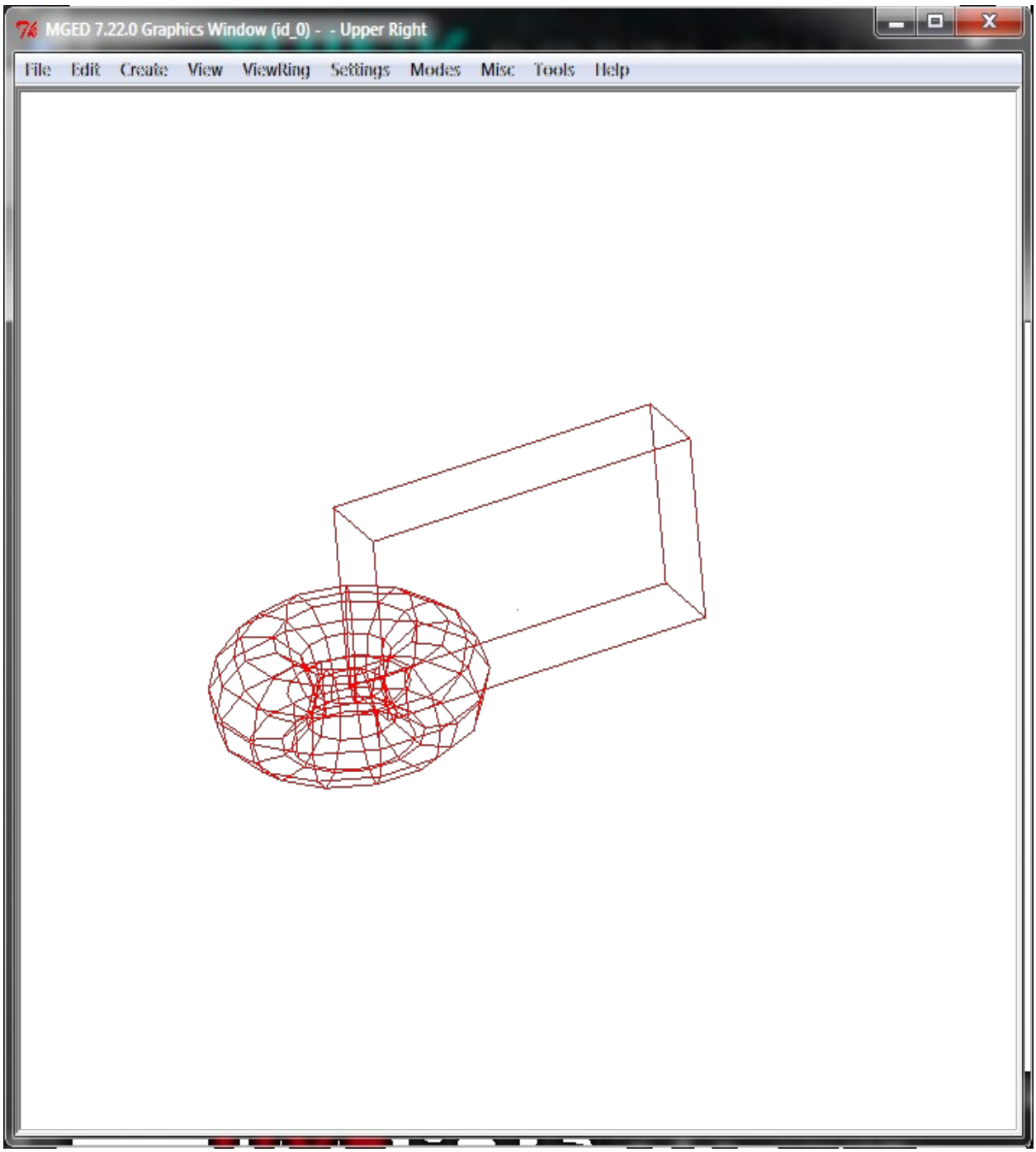

Рисунок 2.10 – Результат роботи програми

2.6 Комбіновані тіла

Робота з комбінованими тілами відбувається наступним чином, спочатку користувач за допомогою командисomb або командиг створює комбінацію, потім командою oed вказує ієрархічний рівень, на якому знаходитися комбіноване тіло. Дана структура містить наступну ієрархію:

Ліве крило.r
Двигун.c
Циліндри.s
Ротор.s
Гвинт.s
Праве крило.r
Двигун.c
Циліндри.s
Ротор.s
Гвинт.s

Відповідно для роботи з крилом необхідно написати:
oed / Ліве крило.r / Двигун.c / Ротор.s
або
oed / Ліве крило.r / Гвинт.s

А щоб працювати з двигуном правого крила:
1 oed / Праве крило.r / Двигун.c / Ротор.s

Під час написання коду були застосовані: команди tra, що дозволяє рух редагованого об'єкта у відносному положенні, команда cp для створення неповної копії об'єкту. Фрагмент коду створення комбінованого об’єкту виглядає наступним чином:

```
mged>comb rod1.c u len0.s u w0.s
mged>cp rod1.c rod2.c
mged>oed / rod2.c/w0.s
Error: Unable to find solid matching path
mged>l rod2.c
mged>closedb
mged>opendb srclg.g
mged>ebase1.srod1.crod2.c
```

```
mged>oed / rod2.c/w0.s
mged>tra 0 8 0
mged>tra 0 5 0
mged>accept
mged>cp rod1.c rod2.c
mged>oed / rod2.c/w0.s
mged>dbversion
mged>dbupgrade
mged>oed / rod2.c/w0.s
```

Для отримання та відображення роботи коду, пропишемо його в командному вікні програми BRL-CAD (рис. 2.11).

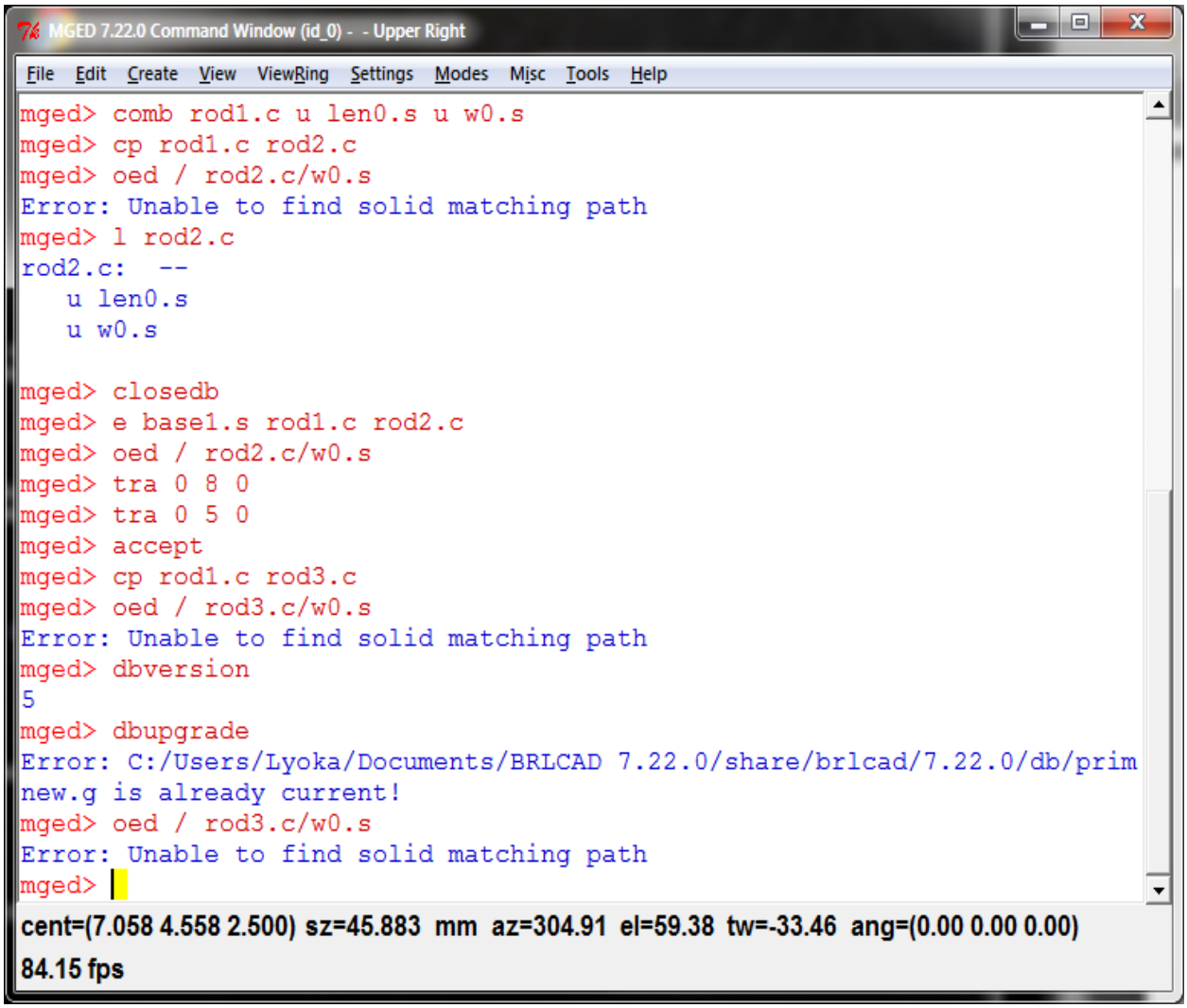

Рисунок 2.11 – Командне вікно програми BRL-CAD

Для перегляду наслідку виконання коду перейдемо до графічного вікна програми BRL-CAD, що зображений на рисунку 2.12. На даному рисунку

показаний результат роботи програмного коду, завдяки якому відбулося копіювання об’єкту тору.

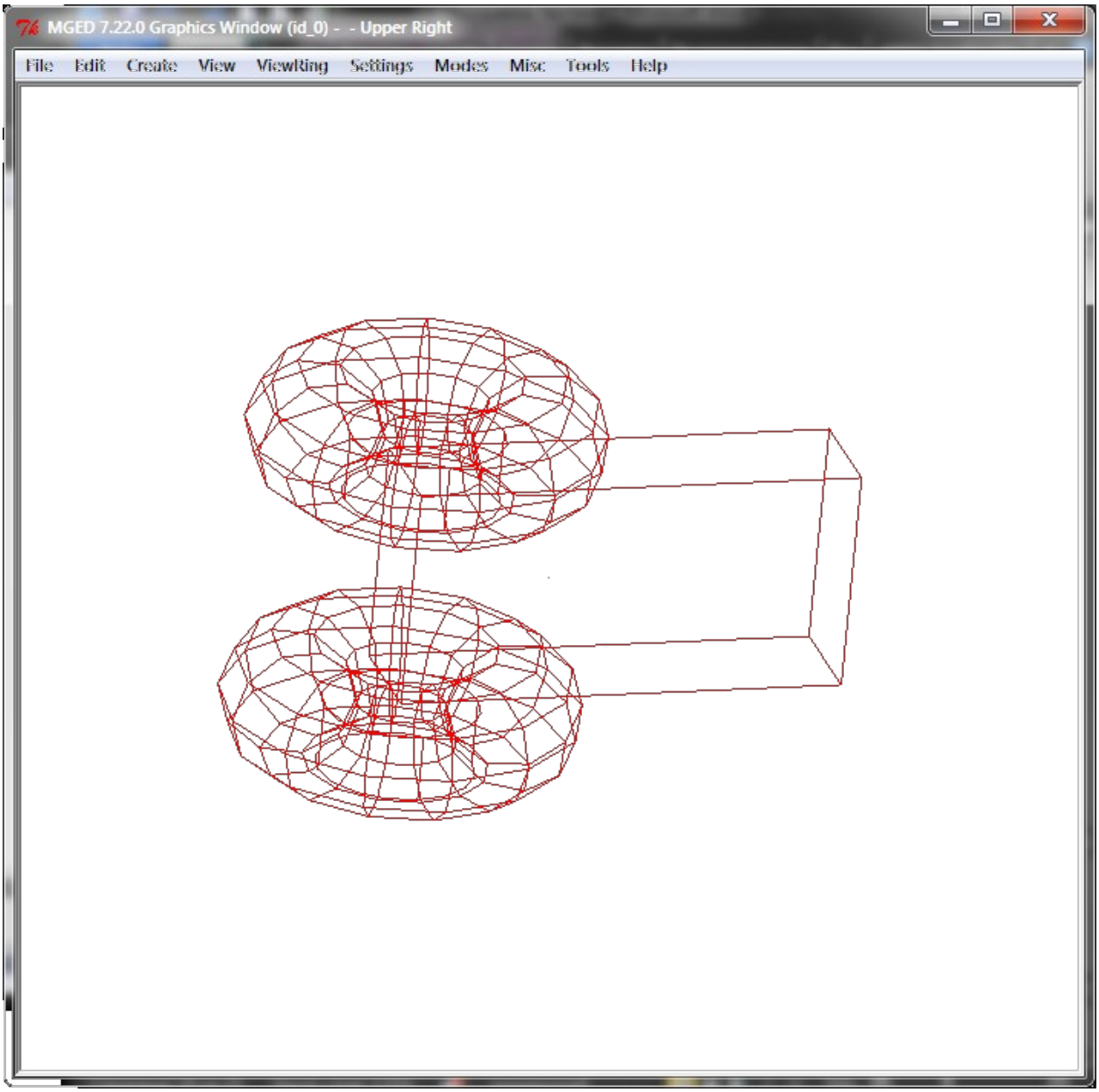

Рисунок 2.12 – Результат роботи застосування коду

Продовжимо роботу з об’єктом. Наступним кроком є створення ще одного колеса, яке буде розміщене на передній частині нашого об’єкта. Під час написання коду були застосовані: команди tra, що дозволяє рух редагованого об'єкта у відносному положенні, команда e для екранний редагування об'єкту, команда oed для редагування матриці (режим редагування об'єкту). Фрагмент коду створення комбінованого об’єкту виглядає наступним чином:

```
mged>closedb
mged> e base1.s rod1.c rod2.c rod2.c
mged>oed / rod2.c/w0.s
mged>tra 0 6.5 0
mged>tra 10 0 0
mged>tra 10 0 0
mged>accept
```

Необхідно зазначити, що після використання команди closedb, користувачеві необхідно знову вказувати ім’я бази даних для її відкриття.

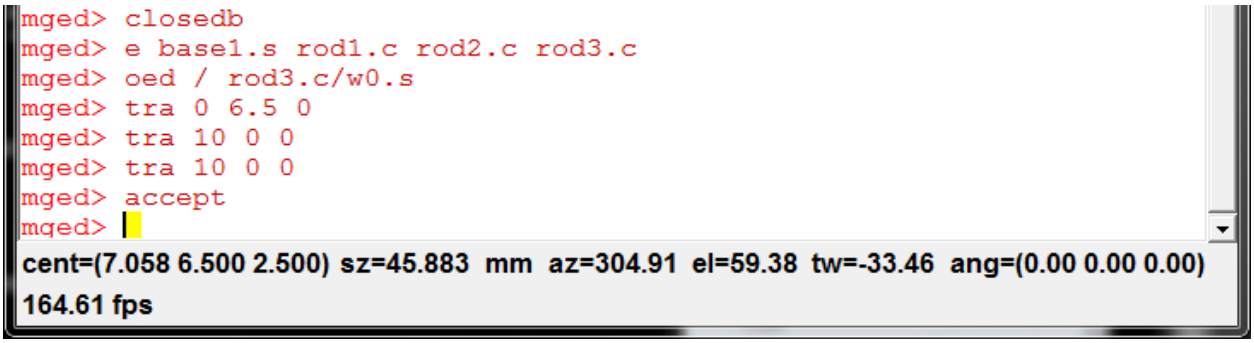

Рисунок 2.13 – Прописаний фрагмент коду в консолі програми

Отже було створено колесо rod1.c з вже існуючого тіла, тобто тора, яке далі було скопійованим з відповідним іменем rod2.c, а потім було переміщеним.

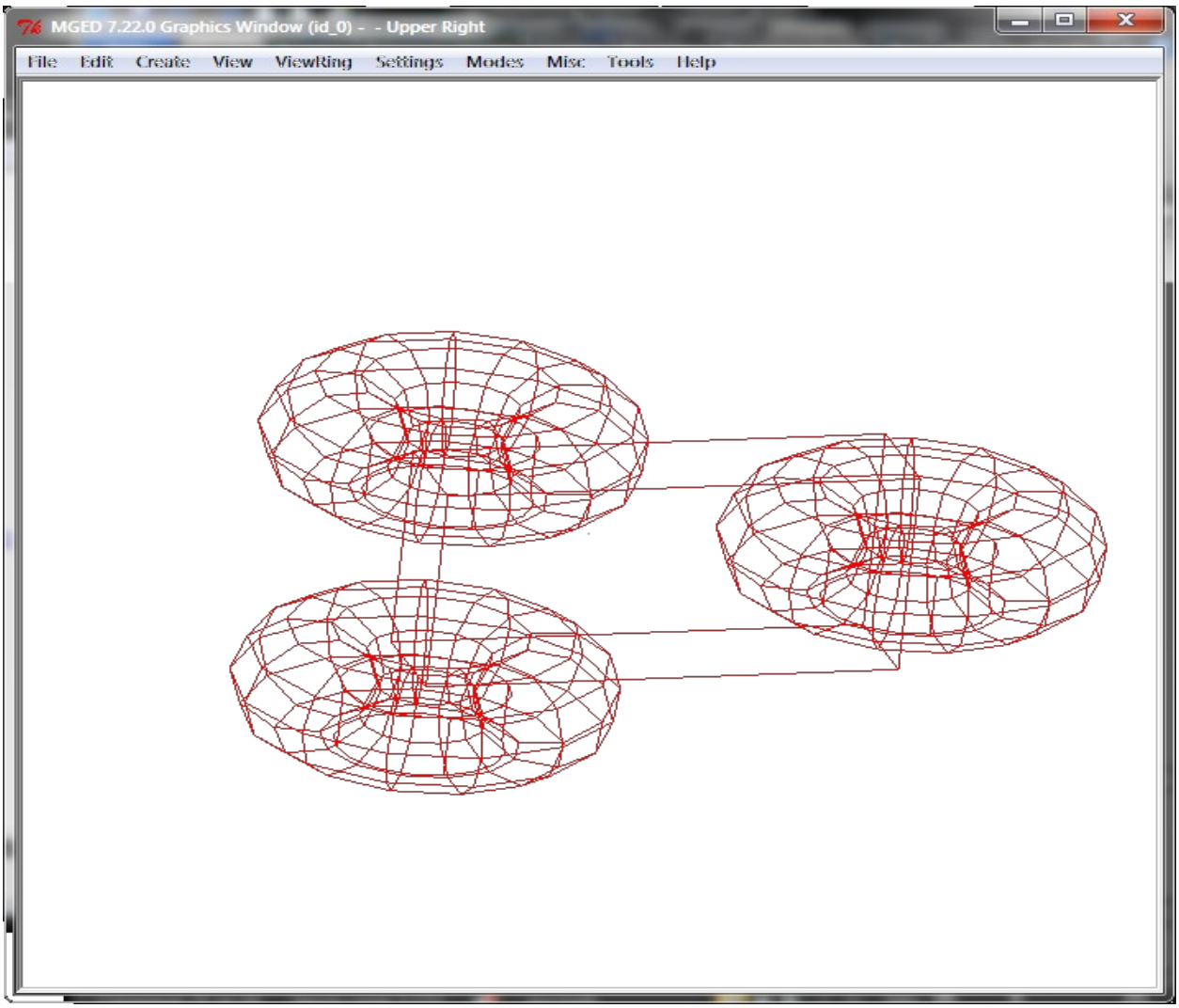

Рисунок 2.14 – Графічне вікно програми з відображенням результату роботи

Останнім етапом виконання роботи є створення на побудованому об’єкті пушки, яка буде розміщена між задніми колесами. Код створення пушки виглядає наступним чином:

```
mged> in
Enter name of solid: stan
Enter solid type: rpc
Enter X, Y, Z of vertex: 0 0 0
Enter X, Y, Z, of vector H: 0 5 0
Enter X, Y, Z, of vector B: 0 0 5
Enter rectangular half-width, r: 3
stan
mged>sedstan
mged>tra 8 2.5 5
```

```
mged> accept
mged> in
Enter name of solid: gun
Enter solid type: rcc
Enter X, Y, Z of vertex: 0 5 0
Enter X, Y, Z of height (H) vector: 10 0 0
Enter radius: 1.5
gun
mged>sed gun
mged>tra 8.5 0 7
mged>tra 0 0 1
mged> rot 0 30 0
mged> accept
```

Для отримання та відображення роботи коду, пропишемо його в командному вікні програми BRL-CAD, рисунок 2.15.

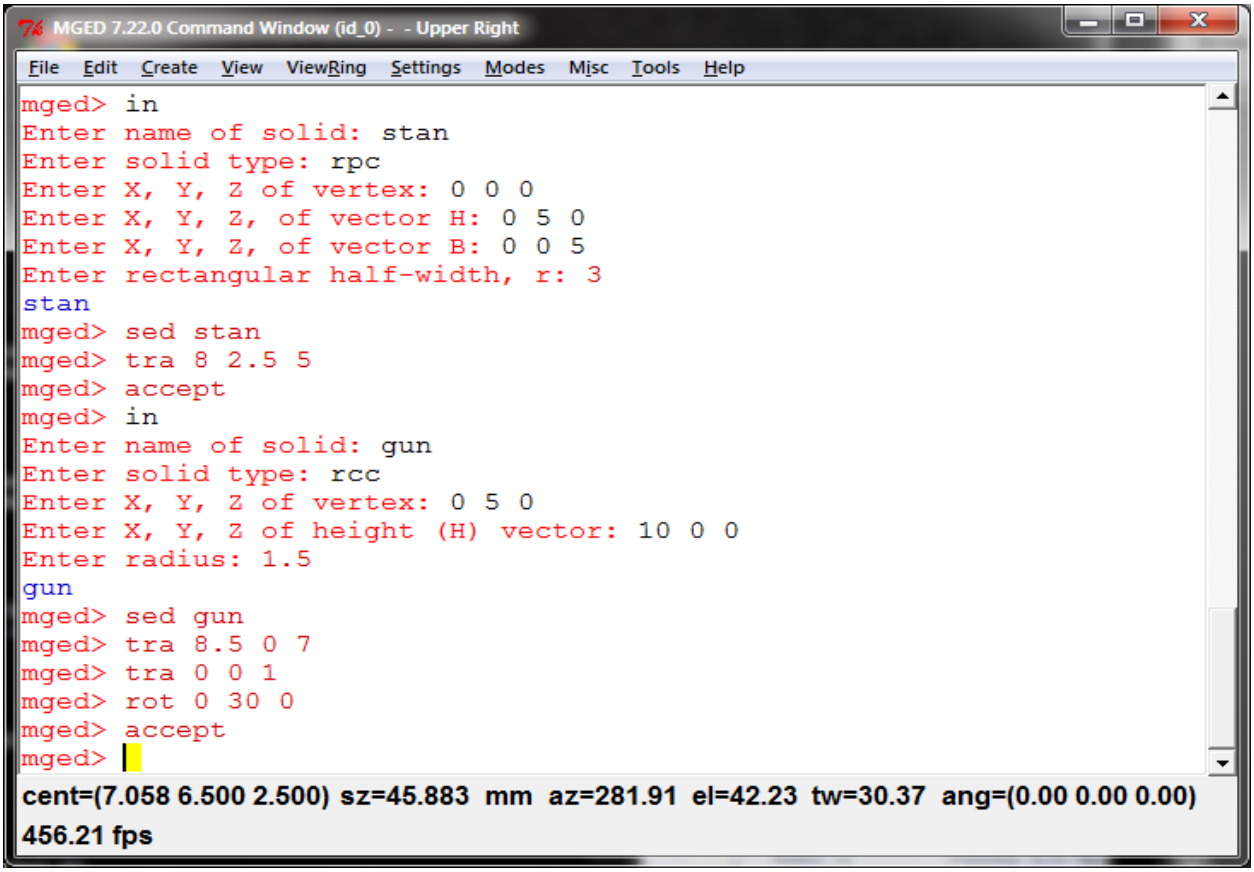

Рисунок 2.15 – Прописаний код в командному вікні

Для перегляду наслідку виконання коду перейдемо до графічного вікна програми BRL-CAD, що зображений на рисунку 2.16. На даному рисунку показаний результат роботи програмного коду, завдяки якому створення пушки об’єкту та її переміщення у центр.

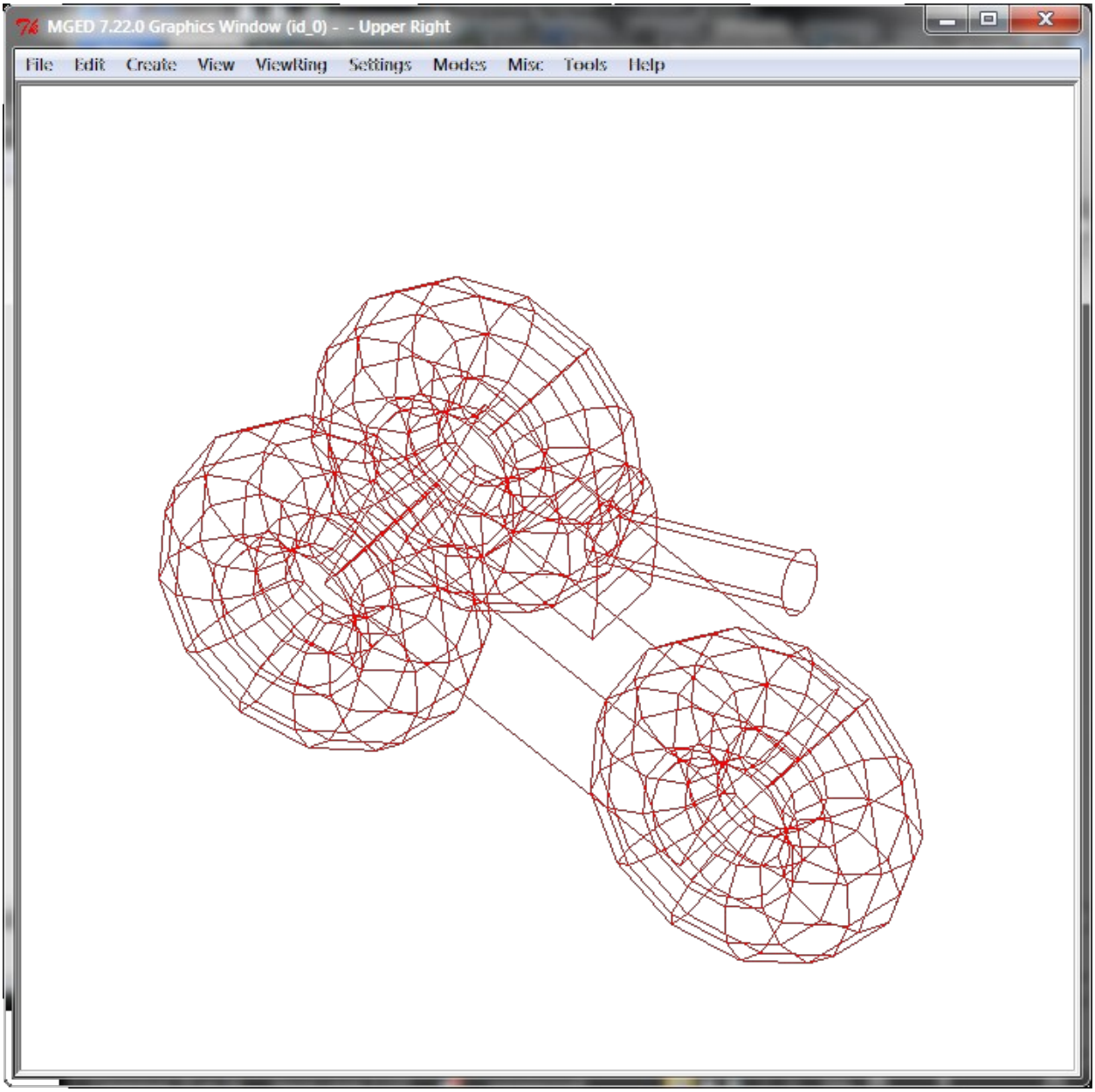

Рисунок 2.16 – Результат роботи коду в програмі BRL-CAD. Відображення створеної пушки на побудованому об’єкті

2.7 Організація рендерінгу та трасування променів

Для повного відображення створеної деталі запустимо процес візуалізації створеного об’єкту в графічному вікні програми BRL-CAD. Створимо процес рендерингу за допомогою застосування команди:

rt -F/dev/X1-s512

Результат виконання команди відображений на рисунку 2.17.

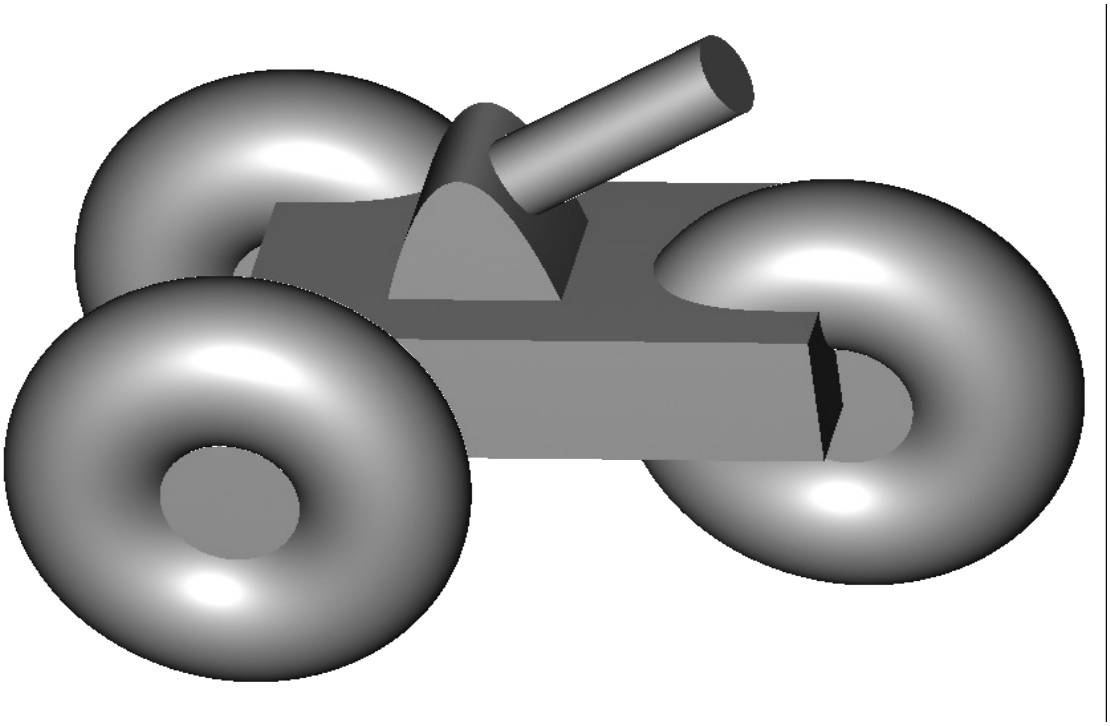

Рисунок 2.17 – Результат роботи процесу рендерингу

2.8 Висновки за розділом

Отже, САПР BRL-CAD є досить простою в застосуванні для досвідченого проектувальника. Проте для початківця – процес її застосування виявиться досить ускладненим.

Наявність двох модулів, що містяться в пакеті допомагає користувачеві системою швидко та зручно створювати та конструювати будь-які об’єкти.

Фундаментальна властивість пакету полягає в його здатності конструювати та аналізувати реалістичні моделі на основі складних об'єктів, що складаються з відносно невеликого набору графічних примітивів.

ВИСНОВКИ

Однією з важливих складових частин САПР – є *машинна графіка*, як сукупність засобів та прийомів, за допомогою яких здійснюється введення, перетворення та виведення з ЕОМ графічної інформації. Машинна графіка – нова *актуальна* галузь проектування та застосування засобів обчислювальної техніки, що інтенсивно розвивається у останній час.

Термін «машинна графіка» означає обробку на ЕОМ графічної інформації, а також виведення результатів у вигляді різних графічних зображень. Графічна інформація – найбільш ємне і наочне уявлення великого обсягу інформації, однак, практичне застосування машинної графіки довгий час стримувалось відсутністю відповідного обладнання та математичного забезпечення.

Відмінною складовою завдань машинної графіки – є обробка ГБД, які, по суті являють собою «звичайні» БД, але в основу яких закладено математичні алгоритми відновлення зображення за сформованими статистичними координаційними даними. Такі можливості є далеко не у кожній САПР, але сучасні тенденції просто вимагають цього. Велика кількість програмних продуктів розроблюється з широким спектром моделюючих характеристик, BRL-CAD – це одна з таких САПР.

В процесі виконання дипломної роботи, було досягнуто її мету, що полягала у створенні, підключенні та оптимізації графічних баз даних, як композиційного компоненту відкритої САПР BRL-CAD.

Об’єктом роботи стали відкриті ГБД, що редагуються засобами BRL-CAD.

У першому розділі дипломної роботи було розглянуто: проблему визначення та історію розвитку БД, їх різновиди; класифікацію за:

а) моделлю даних;

б) середовищем постійного зберігання;

в) вмістом;

г) ступенем розподіленості;

д) об’ємом.

Також у цьому розділі було розглянуто комп’ютерне моделювання складних графічних об’єктів, основоположні принципи відкритого програмного забезпечення та огляд основних САПР-аналогів для роботи з ГБД, а саме: FreeCAD; QCAD COMMUNITY EDITION; SALOME; ELECTRIC VLSI DESIGN SYSTEM; KiCAD; Wings 3D.

У другому розділі (проектна частина) було спроектовано низку діаграм, що всебічно характеризують процес формування ГБД у САПР BRL-CAD, а саме:

а) діаграма варіантів використання;

б) діаграма станів;

в) комунікативна діаграма;

г) діаграма послідовності;

д) діаграма активності;

е) діаграма класів;

ж) діаграма компонентів.

и) діаграма розгортання.

У третьому розділі (програмна частина) розглянуто : роботу з інструментами та утилітами BRL-CAD, створення графічних бібліотек САПР BRL-CAD, її використані команди та можливості, домовленості про імена файлів та імена геометрії. Також у цьому розділі розглянуто: процеси створення простих тіл, логічні операції та операції з комбінованими тілами; організацію рендерінгу та трасування променів.

В цілому, після виконання поданої дипломної роботи, можна зробити наступні висновки:

а) САПР BRL-CAD є досить простою в застосуванні для досвідченого проектувальника, проте для початківця – процес її застосування виявиться досить ускладненим;

б) фундаментальна властивість пакету полягає в його здатності конструювати та аналізувати реалістичні моделі на основі складних об'єктів, які складаються з, відносно, невеликого набору графічних примітивів;

в) потужний бік системи – швидкість засобів візуалізації, трасувальника променів, який є одним з найшвидших серед існуючих.

Остання перевага надає можливості для широкого застосування у різноманітних військових, академічних чи промислових додатках, що включають системи проектування та аналізу автомобілів, механічних вузлів й архітектурних споруд.

Крім того, у дипломній роботі спеціаліста виконано усі додаткові розділи, а саме:

– організаційно-економічне та маркетингове обґрунтування впровадження;

– охорони праці та навколишнього середовища;

– цивільної оборони.

ПЕРЕЛІК ПОСИЛАНЬ

1. Цивільний кодекс України (затв. 16.01.2003, № 435-IV). – К.: Верховна Рада України; док. 435-15, пот. ред.. від 14.06.2007. – 20 с.

2. Закон України «Про авторське право і суміжні права» (введ. в дію Постановою ВР N 3793-XII (3793-12) від 22.12.93, із зм., внесеними від 20.11.2004). – К.: Відомості Верховної Ради, 1994, N 13, ст. 64.

3. Дейт К. Дж. Введение в системы баз данных. 8-е изд. – М.: «Вильямс», 2008. – 508 с.

4. ГОСТ Р ИСО МЭК ТО 10032-2007: Эталонная модель управления данными (идентичен ISO/IEC TR 10032:2003 Information technology — Reference model of data management)

5. ISO/IEC 2382-1:1992. Information technology — Vocabulary — Part 1: Fundamental terms

6. Когаловский М. Р. Энциклопедия технологий баз данных. – М.: Финансы и статистика, 2012. – 460 с.

7. Дейт К. Дж. Введение в системы баз данных = Introduction to Database Systems. — 8-е изд. — М.: Вильямс, 2005. — 1328 с. — ISBN 5-8459-0788-8 (рус.) 0-321-19784-4 (англ.)

8. Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика = Database Systems: A Practical Approach to Design, Implementation, and Management. — 3-е изд. — М.: Вильямс, 2002. — 1436 с. — ISBN 0-201-70857-4

9. Мирошниченко Е. А. К формальному определению понятия «база даных» // Пробл. информатики. 2011. № 2. С. 83-87.

10. Кузнецов С. Д. Основы баз данных. — 2-е изд. — М.: Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2007. — 484 с. — ISBN 978-5-94774-736-2

11. Гарсиа-Молина Г., Ульман Дж., Уидом Дж. Системы баз данных. Полный курс = Database Systems: The Complete Book. — Вильямс, 2002. — 1088 с. — ISBN 5-8459-0384-X

12. Haigh T. How Data Got its Base: Information Storage Software in the 1950s and 1960s // IEEE Annals of the History of Computing. — 2009. — #4 October-December

13. C. J. Date Date on Database: Writings 2000–2006. — Apress, 2006. — 566 с. — ISBN 978-1-59059-746-0, 1-59059-746-X

14. Когаловский М.Р. Перспективные технологии информационных систем. — М.: ДМК Пресс; Компания АйТи, 2002. — 288 с. — ISBN 5-279-02276-4

15. Дж. Ли, Б. Уэр. Трёхмерная графика и анимация. — 2-е изд. — М.: Вильямс, 2002. — 640 с.

16. Д. Херн, М. П. Бейкер. Компьютерная графика и стандарт OpenGL. — 3-е изд. — М., 2005. — 1168 с.

17. Э. Энджел. Интерактивная компьютерная графика. Вводный курс на базе OpenGL. — 2-е изд. — М.: Вильямс, 2001. — 592 с.

18. Г. Снук. 3D-ландшафты в реальном времени на C++ и DirectX 9. — 2-е изд. — М.: Кудиц-пресс, 2007. — 368 с. — ISBN 5-9579-0090-7

19. В. П. Иванов, А. С. Батраков. Трёхмерная компьютерная графика / Под ред. Г. М. Полищука. — М.: Радио и связь, 1995. — 224 с. — ISBN 5-256-01204-5

20. Ларман К. Применение UML 2.0 и шаблонов проектирования. – 3-е изд. / К. Ларман. – М.: Вильямс, 2008. – 736 с.

21. Шмуллер Дж. Освой самостоятельно UML 2.0 за 24 часа. Практическое руководство / Дж. Шмуллер. – М.: Вильямс, 2009. – 416 с.

22. Буч Г. Язык UML. Руководство пользователя. – 5-е изд. / Г. Буч, Дж. Рамбо, А. Джекобсон. – М., СПб.: ДМК Пресс, 2011. – 432 с.

23. Буч Г. UML. Классика CS. – 2-е изд. / Г. Буч, Дж. Рамбо, А. Джекобсон. Пер. с англ.; под общ. ред. О. С. Орлова. – СПб.: Питер, 2010. – 736 с.

24. Гома Х., UML. Проектирование систем реального времени, параллельных и распределенных приложений / Х. Гома.– М.: ДМК Пресс, 2012.– 704 с.

25. Кватрани Т., Rational Rose 2000 и UML. Визуальное моделирование / Т. Кватрани.– М.: ДМК Пресс, 2001.– 176 с.

26. Леоненков А.В., Самоучитель UML / А.В. Леоненков.– СПб.: БХВ-Петербург, 2009. – 304 с.

27. Мацяшек Л. А., Анализ требований и проектирование систем. Разработка информационных систем с использованием UML / Л.А. Мацяшек.– М.: Вильямс, 2012.– 432 с.

28. Рамбо Дж., UML: специальный справочник / Дж. Рамбо, А. Якобсон, Г. Буч.– СПб.: Питер, 2008.– 656 с.

29. Фаулер М., UML. Основы./ М. Фаулер, К. Скотт.– СПб.: Символ-Плюс, 2011.– 192 с.

30. Чмырь И.А. Моделирование систем в среде UML (Unified Modeling Language): Конспект лекций / И.А. Чмырь, М.Ф.Ус.– О.: ОГАХ, 2001.– 70 с.

31. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. – Чинний з 01.01.96. – К.: Держстантдарт, 1995. – 60 с.

ДОДАТОК А

Приклад переведення сформованої ГБД до адаптованого формату BRL (*.g)

```
/*                  G _ T R A N S F E R . C
 * BRL-CAD
/** @file g_transfer.c
 * To compile from an install:
 * gcc -I/usr/brlcad/include/brlcad -L/usr/brlcad/lib -o g_transfer g_transfer.c -lpkg -
lrt -lbu
 */

#include "common.h"

/* system headers */
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include "bio.h"

/* interface headers */
#include "raytrace.h"
#include "bu.h"
#include "pkg.h"

/* used by the client to pass the dbip and opened transfer file
 * descriptor.
 */
typedef struct _my_data_ {
   struct pkg_conn *connection;
   const char *server;
   int port;
} my_data;

/* simple network transport protocol. connection starts with a HELO,
 * then a variable number of GEOM/ARGS messages, then a CIAO to end.
 */
#define MAGIC_ID       "G_TRANSFER"
#define MSG_HELO      1
```

```
#define MSG_ARGS      2
#define MSG_GEOM      3
#define MSG_CIAO      4

/* static size used for temporary string buffers */
#define MAX_DIGITS   32

/* in-memory geometry database filled in by the server as it receives
 * geometry from the client.
 */
struct db_i *DBIP = NULL;

/* used by server to stash what it should shoot at */
int srv_argc = 0;
char **srv_argv = NULL;

/** print a usage statement when invoked with bad, help, or no arguments
 */
void
usage(const char *msg, const char *argv0)
{
   if (msg) {
      bu_log("%s\n", msg);
   }
   bu_log("Usage: %s [-t] [-p#] host gfile [geometry ...]\n\t-p#\tport number to send to (default 2000)\n\thost\thostname or IP address of receiving server\n\tgfile\tBRL-CAD .g database file\n\tgeometry\tname(s) of geometry to send (OPTIONAL)\n", argv0 ? argv0 : "g_transfer");
   bu_exit(1, "Usage: %s -r [-p#]\n\t-p#\tport number to listen on (default 2000)\n", argv0 ? argv0 : "g_transfer");
}

void
validate_port(int port) {
   if (port < 0)
      bu_exit(EXIT_FAILURE, "Invalid negative port range\n");
}
```

```
int
hit(struct application *UNUSED(ap), struct partition *UNUSED(p), struct seg *UN-
USED(s))
{
   bu_log("HIT!\n");
   return 0;
}

int
miss(struct application *UNUSED(ap))
{
   bu_log("MISSED!\n");
   return 0;
}

void
do_something() {
   /* shoot a ray at some geometry just to show that we can */
   struct application ap;
   struct rt_i *rtip;
   int i;

   if (!DBIP) {
      return;
   }

   RT_APPLICATION_INIT(&ap);
   rtip = rt_new_rti(DBIP); /* clone dbip */
   if (!rtip) {
      bu_log("Unable to create a database instance off of the raytrace instance\n");
      return;
   }
   rt_ck(rtip); /* gratuitous sanity check, test for corruption */
   ap.a_rt_i = rtip;
   ap.a_hit = hit;
   ap.a_miss = miss;
   VSET(ap.a_ray.r_pt, 0, 0, 10000);
   VSET(ap.a_ray.r_dir, 0, 0, -1);
```

```
    /* shoot at any geometry specified */
    for (i = 0; i < srv_argc; i++) {
        if (rt_gettree(rtip, srv_argv[i]) != 0) {
            bu_log("Unable to validate %s for raytracing\n", srv_argv[i]);
            continue;
        }
        rt_prep(rtip);
        bu_log("Shooting at %s from (0, 0, 10000) in the (0, 0, -1) direction\n",
srv_argv[i]);
        (void)rt_shootray(&ap);
        rt_clean(rtip);
    }
    rt_free_rti(rtip);
}

void
server_helo(struct pkg_conn *UNUSED(connection), char *buf)
{
    /* should not encounter since we listened for it specifically
     * before beginning processing of packets.
     */
    bu_log("Unexpected HELO encountered\n");
    free(buf);
}

void
server_args(struct pkg_conn *UNUSED(connection), char *buf)
{
    /* updates the srv_argc and srv_argv application globals used to
     * show that we can shoot at geometry in-memory.
     */
    srv_argc++;
    if (!srv_argv) {
        srv_argv = bu_calloc(1, srv_argc * sizeof(char *), "server_args() srv_argv cal-
loc");
    } else {
        srv_argv = bu_realloc(srv_argv, srv_argc * sizeof(char *), "server_args()
srv_argv realloc");
    }
```

```
    srv_argv[srv_argc - 1] = bu_calloc(1, strlen(buf)+1, "server_args() srv_argv[] cal-
loc");
    bu_strlcpy(srv_argv[srv_argc - 1], buf, strlen(buf)+1);

    bu_log("Planning to shoot at %s\n", buf);

    free(buf);
}

void
server_geom(struct pkg_conn *UNUSED(connection), char *buf)
{
    struct bu_external ext;
    struct db5_raw_internal raw;
    int flags;

    if (DBIP == NULL) {
        /* first geometry received, initialize */
        DBIP = db_open_inmem();
    }

    if (db5_get_raw_internal_ptr(&raw, (const unsigned char *)buf) == NULL) {
        bu_log("Corrupted serialized geometry?  Could not deserialize.\n");
        free(buf);
        return;
    }

    /* initialize an external structure since the data seems valid and add/export
     * it to the directory.
     */
    BU_EXTERNAL_INIT(&ext);
    ext.ext_buf = (uint8_t *)buf;
    ext.ext_nbytes = raw.object_length;
    flags = db_flags_raw_internal(&raw) | RT_DIR_INMEM;
     wdb_export_external(DBIP->dbi_wdbp, &ext, (const char *)raw.name.ext_buf,
flags, raw.minor_type);

     bu_log("Received %s (MAJOR=%d, MINOR=%d)\n", raw.name.ext_buf, raw.-
major_type, raw.minor_type);
```

```
}

void
server_ciao(struct pkg_conn *UNUSED(connection), char *buf)
{
    bu_log("CIAO encountered\n");

    /* shoot some rays just to show that we can if server was
     * invoked with specific geometry.
     */
    do_something();

    if (DBIP != NULL) {
        /* uncomment to avoid an in-mem dbip close bug */
        /* DBIP->dbi_fp = fopen("/dev/null", "rb");*/
        db_close(DBIP);
        DBIP = NULL;
    }

    free(buf);
}

/** start up a server that listens for a single client.
 */
void
run_server(int port) {
    struct pkg_conn *client;
    int netfd;
    char portname[MAX_DIGITS] = {0};
    int pkg_result  = 0;
    char *title;

    struct pkg_switch callbacks[] = {
        {MSG_HELO, server_helo, "HELO", NULL},
        {MSG_ARGS, server_args, "ARGS", NULL},
        {MSG_GEOM, server_geom, "GEOM", NULL},
        {MSG_CIAO, server_ciao, "CIAO", NULL},
        {0, 0, NULL, NULL}
    };
```

```
    validate_port(port);

    /* start up the server on the given port */
    snprintf(portname, MAX_DIGITS - 1, "%d", port);
    netfd = pkg_permserver(portname, "tcp", 0, 0);
    if (netfd < 0)
        bu_exit(EXIT_FAILURE, "Unable to start the server");

    /* listen for a good client indefinitely */
    do {
        client = pkg_getclient(netfd, callbacks, NULL, 0);
        if (client == PKC_NULL) {
            bu_log("Connection seems to be busy, waiting...\n");
            sleep(10);
            continue;
        } else if (client == PKC_ERROR) {
            bu_log("Fatal error accepting client connection.\n");
            pkg_close(client);
            client = PKC_NULL;
            continue;
        }

        /* got a connection, process it */
        title = pkg_bwaitfor (MSG_HELO, client);
        if (title == NULL) {
            bu_log("Failed to process the client connection, still waiting\n");
            pkg_close(client);
            client = PKC_NULL;
        } else {
            /* validate magic header */
            if (!BU_STR_EQUAL(title, MAGIC_ID)) {
          bu_log("Bizarre corruption, received a HELO without at matching MAGIC
ID!\n");
          pkg_close(client);
          client = PKC_NULL;
            } else {
          title += strlen(MAGIC_ID) + 1;
```

```
        bu_log("Preparing to receive data for geometry from a database with the fol-
lowing title:\n%s\n", title);
            }
        }
    } while (client == PKC_NULL);

    /* read from the connection */
    bu_log("Processing objects from client\n");
    do {
        /* process packets potentially received in a processing callback */
        pkg_result = pkg_process(client);
        if (pkg_result < 0) {
            bu_log("Unable to process packets? Wierd.\n");
        } else {
            bu_log("Processed %d packet%s\n", pkg_result, pkg_result == 1 ? "" : "s");
        }

        /* suck in data from the network */
        pkg_result = pkg_suckin(client);
        if (pkg_result < 0) {
            bu_log("Seemed to have trouble sucking in packets.\n");
            break;
        } else if (pkg_result == 0) {
            bu_log("Client closed the connection.\n");
            break;
        }

        /* process packets received */
        pkg_result = pkg_process(client);
        if (pkg_result < 0) {
            bu_log("Unable to process packets? Wierd.\n");
        } else {
            bu_log("Processed %d packet%s\n", pkg_result, pkg_result == 1 ? "" : "s");
        }
    } while (client != NULL);

    /* shut down the server */
    pkg_close(client);
}
```

```
/**
 * base routine that the client uses to send an object to the server.
 * this is the hook callback function for both the primitives and
 * combinations encountered during a db_functree() traversal.
 *
 * returns 0 if unsuccessful
 * returns 1 if successful
 */
void
send_to_server(struct db_i *dbip, struct directory *dp, genptr_t connection)
{
   my_data *stash;
   struct bu_external ext;
   int bytes_sent = 0;

   RT_CK_DBI(dbip);
   RT_CK_DIR(dp);

   stash = (my_data *)connection;

   if (db_get_external(&ext, dp, dbip) < 0) {
      bu_log("Failed to read %s, skipping\n", dp->d_namep);
      return;
   }

   /* send the external representation over the wire */
   bu_log("Sending %s\n", dp->d_namep);

   /* pad the data with the length in ascii for convenience */
    bytes_sent = pkg_send(MSG_GEOM, (const char *)ext.ext_buf, ext.ext_nbytes,
stash->connection);
   if (bytes_sent < 0) {
      pkg_close(stash->connection);
      bu_log("Unable to successfully send %s to %s, port %d.\n", dp->d_namep,
stash->server, stash->port);
      return;
   }

   /* our responsibility to free the stuff we got */
```

```
    bu_free_external(&ext);
}

/**
 * start up a client that connects to the given server, and sends
 * serialized .g data.  if the user specified geometry, only that
 * geometry is sent via send_to_server().
 */
void
run_client(const char *server, int port, struct db_i *dbip, int geomc, const char
**geomv)
{
    my_data stash;
    int i;
    struct directory *dp;
    char s_port[MAX_DIGITS] = {0};
    int bytes_sent = 0;

    RT_CK_DBI(dbip);
    /* open a connection to the server */
    validate_port(port);

    snprintf(s_port, MAX_DIGITS - 1, "%d", port);
        stash.connection = pkg_open(server, s_port, "tcp", NULL, NULL, NULL,
NULL);
    if (stash.connection == PKC_ERROR) {
        bu_log("Connection to %s, port %d, failed.\n", server, port);
        bu_exit(EXIT_FAILURE, "ERROR: Unable to open a connection to the
server\n");
    }
    stash.server = server;
    stash.port = port;

    /* let the server know we're cool.  also, send the database title
     * along with the MAGIC ident just because we can.
     */
       bytes_sent = pkg_2send(MSG_HELO, MAGIC_ID, strlen(MAGIC_ID) + 1,
dbip->dbi_title, strlen(dbip->dbi_title), stash.connection);
    if (bytes_sent < 0) {
```

```
        pkg_close(stash.connection);
        bu_log("Connection to %s, port %d, seems faulty.\n", server, port);
        bu_exit(EXIT_FAILURE, "ERROR: Unable to communicate with the server\
n");
    }

    bu_log("Database title is:\n%s\n", dbip->dbi_title);
    bu_log("Units: %s\n", bu_units_string(dbip->dbi_local2base));

    /* send geometry to the server */
    if (geomc > 0) {
        /* geometry was specified. look it up and process the
         * hierarchy using db_functree() where all combinations and
         * primitives are sent that get encountered.
         */
        for (i = 0; i < geomc; i++) {
            /* send the geometry as an ARGS packet so the server can
             * know what to shoot at.
             */
            bytes_sent = pkg_send(MSG_ARGS, geomv[i], strlen(geomv[i]) + 1, stash.-
connection);
            if (bytes_sent < 0) {
          pkg_close(stash.connection);
          bu_log("Unable to request server shot at %s\n", geomv[i]);
          bu_exit(EXIT_FAILURE, "ERROR: Unable to communicate request to
server\n");
            }

            dp = db_lookup(dbip, geomv[i], LOOKUP_NOISY);
            if (dp == RT_DIR_NULL) {
          pkg_close(stash.connection);
          bu_log("Unable to lookup %s\n", geomv[i]);
          bu_exit(EXIT_FAILURE, "ERROR: requested geometry could not be found\
n");
            }
              db_functree(dbip, dp, send_to_server, send_to_server, &rt_uniresource,
(genptr_t)&stash);
        }
    } else {
```

```
        /* no geometry was specified so traverse the array of linked
         * lists contained in the database instance and send
         * everything.
         */
        FOR_ALL_DIRECTORY_START(dp, dbip) {
            send_to_server(dbip, dp, (genptr_t)&stash);
        } FOR_ALL_DIRECTORY_END;
    }

    /* let the server know we're done.  not necessary, but polite. */
    bytes_sent = pkg_send(MSG_CIAO, "BYE", 4, stash.connection);
    if (bytes_sent < 0) {
        bu_log("Unable to cleanly disconnect from %s, port %d.\n", server, port);
    }

    /* flush output and close */
    pkg_close(stash.connection);

    return;
}

int
main(int argc, char *argv[]) {
    const char * const argv0 = argv[0];
    int c;
    int server = 0; /* not a server by default */
    int port = 2000;

    /* client stuff */
    const char *server_name = NULL;
    const char *geometry_file = NULL;
    const char ** geometry = NULL;
    int ngeometry = 0;
    struct db_i *dbip = NULL;

    if (argc < 2) {
        usage("ERROR: Missing arguments", argv[0]);
    }
```

```
/* process the command-line arguments after the application name */
while ((c = bu_getopt(argc, argv, "tTrRp:P:hH")) != -1) {
    switch (c) {
        case 't':
        case 'T':
      /* sending */
      server = 0;
      break;
        case 'r':
        case 'R':
      /* receiving */
      server = 1;
      break;
        case 'p':
        case 'P':
      port = atoi(bu_optarg);
      break;
        case 'h':
        case 'H':
      /* help */
      usage(NULL, argv0);
      break;
        default:
      usage("ERROR: Unknown argument", argv0);
    }
}

argc -= bu_optind;
argv += bu_optind;

if (server) {
    if (argc > 0) {
        usage("ERROR: Unexpected extra arguments", argv0);
    }

    /* mark the database as in-memory only */
    /* XXX = wdb_dbopen(dbip, RT_WDB_TYPE_DB_INMEM); */

    /* ignore broken pipes */
```

```
#ifdef SIGPIPE
    (void)signal(SIGPIPE, SIG_IGN);
#endif

    /* fire up the server */
    bu_log("Listening on port %d\n", port);
    run_server(port);

    return 0;
}

/* prep up the client */
if (argc < 1) {
    usage("ERROR: Missing hostname and geometry file arguments", argv0);
} else if (argc < 2) {
    usage("ERROR: Missing geometry file argument", argv0);
} else {
    geometry = (const char **)(argv + 2);
    ngeometry = argc - 2;
}

server_name = *argv++;
geometry_file = *argv++;

/* make sure the geometry file exists */
if (!bu_file_exists(geometry_file)) {
    bu_log("Geometry file does not exist: %s\n", geometry_file);
    bu_exit(EXIT_FAILURE, "Need a BRL-CAD .g geometry database file\n");
}

/* XXX fixed in latest db_open(), but call for now just in case
   until 7.8.0 release */
rt_init_resource(&rt_uniresource, 0, NULL);

/* make sure the geometry file is a geometry database, get a
 * database instance pointer.
 */
dbip = db_open(geometry_file, "r");
if (dbip == DBI_NULL) {
```

```
        bu_log("Cannot open %s\n", geometry_file);
        perror(argv0);
        bu_exit(EXIT_FAILURE, "Need a geometry file");
    }

    /* load the database directory into memory */
    if (db_dirbuild(dbip) < 0) {
        db_close(dbip);
        bu_log("Unable to load the database directory for file: %s\n", geometry_file);
        bu_exit(EXIT_FAILURE, "Can't read geometry file");
    }

    /* fire up the client */
    bu_log("Connecting to %s, port %d\n", server_name, port);
    run_client(server_name, port, dbip, ngeometry, geometry);

    /* done with the database */
    db_close(dbip);

    return 0;
}

/*
 * Local Variables:
 * mode: C
 * tab-width: 8
 * indent-tabs-mode: t
 * c-file-style: "stroustrup"
 * End:
 * ex: shiftwidth=4 tabstop=8
 */
```

ДОДАТОК Б

Приклад формування ГБД у вигляді фасетної моделі

```
/*                  G - X X X _ F A C E T S . C
 * BRL-CAD
/** @file conv/g-xxx_facets.c

#include "common.h"

#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <string.h>

#include "vmath.h"
#include "nmg.h"
#include "rtgeom.h"
#include "raytrace.h"


#define V3ARGSIN(a)       (a)[X]/25.4, (a)[Y]/25.4, (a)[Z]/25.4
#define VSETIN( a, b )    {\
        (a)[X] = (b)[X]/25.4; \
        (a)[Y] = (b)[Y]/25.4; \
        (a)[Z] = (b)[Z]/25.4; \
    }

extern union tree *do_region_end(struct db_tree_state *tsp, const struct
db_full_path *pathp, union tree *curtree, genptr_t client_data);

extern double nmg_eue_dist;             /* from nmg_plot.c */

static char    usage[] = "\
Usage: %s [-v][-xX lvl][-a abs_tess_tol (default: 0.0)][-r rel_tess_tol (default:
0.01)]\n\
  [-n norm_tess_tol (default: 0.0)][-D dist_calc_tol (default: 0.005)]\n\
   -o output_file_name brlcad_db.g object(s)\n";
```

```
static int      NMG_debug;      /* saved arg of -X, for longjmp handling */
static int      verbose;
static struct db_i          *dbip;
static struct rt_tess_tol   ttol;   /* tesselation tolerance in mm */
static struct bn_tol        tol;    /* calculation tolerance */
static struct model         *the_model;

static struct db_tree_state         tree_state;   /* includes tol & model */

static int              regions_tried = 0;
static int              regions_converted = 0;
static int              regions_written = 0;
static unsigned int tot_polygons = 0;

/*
 *              M A I N
 */
int
main(int argc, char **argv)
{
    int  c;
    double              percent;
    int         i;

    bu_setlinebuf( stderr );

    tree_state = rt_initial_tree_state;     /* struct copy */
    tree_state.ts_tol = &tol;
    tree_state.ts_ttol = &ttol;
    tree_state.ts_m = &the_model;

    /* Set up tesselation tolerance defaults */
    ttol.magic = RT_TESS_TOL_MAGIC;
    /* Defaults, updated by command line options. */
    ttol.abs = 0.0;
    ttol.rel = 0.01;
    ttol.norm = 0.0;

    /* Set up calculation tolerance defaults */
```

```
/* FIXME: These need to be improved */
tol.magic = BN_TOL_MAGIC;
tol.dist = 0.0005;
tol.dist_sq = tol.dist * tol.dist;
tol.perp = 1e-5;
tol.para = 1 - tol.perp;

/* init resources we might need */
rt_init_resource( &rt_uniresource, 0, NULL );

/* make empty NMG model */
the_model = nmg_mm();

/* Get command line arguments. */
while ((c = bu_getopt(argc, argv, "r:a:n:o:vx:D:X:")) != -1) {
   switch (c) {
      case 'r':             /* Relative tolerance. */
     ttol.rel = atof(bu_optarg);
     break;
      case 'a':             /* Absolute tolerance. */
     ttol.abs = atof(bu_optarg);
     ttol.rel = 0.0;
     break;
      case 'n':             /* Surface normal tolerance. */
     ttol.norm = atof(bu_optarg);
     ttol.rel = 0.0;
     break;
      case 'o':             /* Output file name. */
     /* grab output file name */
     break;
      case 'v':
     verbose++;
     break;
      case 'x':
     sscanf( bu_optarg, "%x", (unsigned int *)&rt_g.debug );
     break;
      case 'D':
     tol.dist = atof(bu_optarg);
     tol.dist_sq = tol.dist * tol.dist;
```

```
      rt_pr_tol( &tol );
      break;
        case 'X':
      sscanf( bu_optarg, "%x", (unsigned int *)&rt_g.NMG_debug );
      NMG_debug = rt_g.NMG_debug;
      break;
        default:
      bu_exit(1, usage, argv[0]);
      break;
    }
}

if (bu_optind+1 >= argc) {
    bu_exit(1, usage, argv[0]);
}

/* Open output file */


/* Open BRL-CAD database */
argc -= bu_optind;
argv += bu_optind;
if ((dbip = db_open(argv[0], "r")) == DBI_NULL) {
    perror(argv[0]);
    bu_exit(1, "ERROR: Unable to open geometry file (%s)\n", argv[0]);
}
if ( db_dirbuild( dbip ) ) {
    bu_exit(1, "db_dirbuild failed\n" );
}

BN_CK_TOL(tree_state.ts_tol);
RT_CK_TESS_TOL(tree_state.ts_ttol);

if ( verbose ) {
    bu_log( "Model: %s\n", argv[0] );
    bu_log( "Objects:" );
    for ( i=1; i<argc; i++ )
        bu_log( " %s", argv[i] );
```

```
        bu_log( "\nTesselation tolerances:\n\tabs = %g mm\n\trel = %g\n\tnorm = %g\
n",
          tree_state.ts_ttol->abs, tree_state.ts_ttol->rel, tree_state.ts_ttol->norm );
        bu_log( "Calculational tolerances:\n\tdist = %g mm perp = %g\n",
          tree_state.ts_tol->dist, tree_state.ts_tol->perp );
    }

    /* Walk indicated tree(s).  Each region will be output separately */
    (void) db_walk_tree(dbip, argc-1, (const char **)(argv+1),
                        1,                  /* ncpu */
                        &tree_state,
                        0,                  /* take all regions */
                        do_region_end,
                        nmg_booltree_leaf_tess,
                        (genptr_t)NULL);    /* in librt/nmg_bool.c */

    percent = 0;
    if (regions_tried>0) {
        percent = ((double)regions_converted * 100) / regions_tried;
        bu_log("Tried %d regions, %d converted to NMG's successfully.  %g%%\n",
               regions_tried, regions_converted, percent);
    }
    percent = 0;

    if ( regions_tried > 0 ) {
        percent = ((double)regions_written * 100) / regions_tried;
        bu_log( "                  %d triangulated successfully. %g%%\n",
          regions_written, percent );
    }

    bu_log( "%ld triangles written\n", tot_polygons );

    bu_log( "Tesselation parameters used:\n");
    bu_log( "  abs  [-a]    %g\n", ttol.abs );
    bu_log( "  rel  [-r]    %g\n", ttol.rel );
    bu_log( "  norm [-n]    %g\n", ttol.norm );
    bu_log( "  dist [-D]    %g\n", tol.dist );

    /* Release dynamic storage */
```

```
    nmg_km(the_model);
    rt_vlist_cleanup();
    db_close(dbip);

    return 0;
}

/* routine to output the facetted NMG representation of a BRL-CAD region */
static void
output_nmg(struct nmgregion *r, const struct db_full_path *pathp, int UNUSED(region_id), int UNUSED(material_id))
{
    struct model *m;
    struct shell *s;
    struct vertex *v;
    char *region_name;
    int region_polys=0;

    NMG_CK_REGION( r );
    RT_CK_FULL_PATH(pathp);

    region_name = db_path_to_string( pathp );

    m = r->m_p;
    NMG_CK_MODEL( m );

    /* triangulate model */
    nmg_triangulate_model( m, &tol );

    /* Output triangles */
    if ( verbose ) {
        printf( "Convert these triangles to your format for region %s\n", region_name );
    } else {
        printf( "Converted %s\n", region_name );
    }
    for ( BU_LIST_FOR( s, shell, &r->s_hd ) )
    {
        struct faceuse *fu;
```

```
NMG_CK_SHELL( s );

for ( BU_LIST_FOR( fu, faceuse, &s->fu_hd ) )
{
   struct loopuse *lu;
   vect_t facet_normal;

   NMG_CK_FACEUSE( fu );

   if ( fu->orientation != OT_SAME )
  continue;

   /* Grab the face normal if needed */
   NMG_GET_FU_NORMAL( facet_normal, fu);

   for ( BU_LIST_FOR( lu, loopuse, &fu->lu_hd ) )
   {
  struct edgeuse *eu;

  NMG_CK_LOOPUSE( lu );

  if ( BU_LIST_FIRST_MAGIC ( &lu->down_hd ) !=
NMG_EDGEUSE_MAGIC )
     continue;

  /* loop through the edges in this loop (facet) */
  if ( verbose ) {
     printf( "\tfacet:\n" );
  }
  for ( BU_LIST_FOR( eu, edgeuse, &lu->down_hd ) )
  {
     NMG_CK_EDGEUSE( eu );

     v = eu->vu_p->v_p;
     NMG_CK_VERTEX( v );
     if ( verbose ) {
          printf( "\t\t(%g %g %g)\n", V3ARGS( v->vg_p->coord ) );
     }
  }
```

```
            tot_polygons++;
            region_polys++;
              }
            }
        }

        bu_free( region_name, "region name" );
}


static void
process_triangulation(struct nmgregion *r, const struct db_full_path *pathp, struct
db_tree_state *tsp)
{
    if (!BU_SETJUMP) {
        /* try */

        /* Write the facetized region to the output file */
        output_nmg( r, pathp, tsp->ts_regionid, tsp->ts_gmater );

    } else {
        /* catch */

        char *sofar;

        sofar = db_path_to_string(pathp);
        bu_log( "FAILED in triangulator: %s\n", sofar );
        bu_free( (char *)sofar, "sofar" );

        /* Sometimes the NMG library adds debugging bits when
         * it detects an internal error, before bombing out.
         */
        rt_g.NMG_debug = NMG_debug;        /* restore mode */

        /* Release any intersector 2d tables */
        nmg_isect2d_final_cleanup();

        /* Get rid of (m)any other intermediate structures */
        if ( (*tsp->ts_m)->magic == NMG_MODEL_MAGIC ) {
```

```
            nmg_km(*tsp->ts_m);
        } else {
            bu_log("WARNING: tsp->ts_m pointer corrupted, ignoring it.\n");
        }

        /* Now, make a new, clean model structure for next pass. */
        *tsp->ts_m = nmg_mm();
    }  BU_UNSETJUMP;
}

static union tree *
process_boolean(union tree *curtree, struct db_tree_state *tsp, const struct
db_full_path *pathp)
{
    union tree *ret_tree = TREE_NULL;

    /* Begin bomb protection */
    if ( !BU_SETJUMP ) {
        /* try */

        (void)nmg_model_fuse(*tsp->ts_m, tsp->ts_tol);
        ret_tree = nmg_booltree_evaluate(curtree, tsp->ts_tol, &rt_uniresource);

    } else  {
        /* catch */
        char *name = db_path_to_string( pathp );

        /* Error, bail out */
        bu_log( "conversion of %s FAILED!\n", name );

        /* Sometimes the NMG library adds debugging bits when
         * it detects an internal error, before before bombing out.
         */
        rt_g.NMG_debug = NMG_debug;/* restore mode */

        /* Release any intersector 2d tables */
        nmg_isect2d_final_cleanup();

        /* Release the tree memory & input regions */
```

```
        db_free_tree(curtree, &rt_uniresource);/* Does an nmg_kr() */

        /* Get rid of (m)any other intermediate structures */
        if ( (*tsp->ts_m)->magic == NMG_MODEL_MAGIC ) {
            nmg_km(*tsp->ts_m);
        } else {
            bu_log("WARNING: tsp->ts_m pointer corrupted, ignoring it.\n");
        }

        bu_free( name, "db_path_to_string" );
        /* Now, make a new, clean model structure for next pass. */
        *tsp->ts_m = nmg_mm();
    } BU_UNSETJUMP;/* Relinquish the protection */

    return ret_tree;
}

/*
 *              D O _ R E G I O N _ E N D
 *
 *  Called from db_walk_tree().
 *
 *  This routine must be prepared to run in parallel.
 */
union tree *do_region_end(struct db_tree_state *tsp, const struct db_full_path
*pathp, union tree *curtree, genptr_t UNUSED(client_data))
{
    union tree          *ret_tree;
    struct bu_list              vhead;
    struct nmgregion        *r;

    RT_CK_FULL_PATH(pathp);
    RT_CK_TREE(curtree);
    RT_CK_TESS_TOL(tsp->ts_ttol);
    BN_CK_TOL(tsp->ts_tol);
    NMG_CK_MODEL(*tsp->ts_m);

    BU_LIST_INIT(&vhead);
```

```
{
   char     *sofar = db_path_to_string(pathp);
   bu_log("\ndo_region_end(%d %d%%) %s\n",
         regions_tried,
         regions_tried>0 ? (regions_converted * 100) / regions_tried : 0,
         sofar);
   bu_free(sofar, "path string");
}

if (curtree->tr_op == OP_NOP)
   return curtree;

regions_tried++;

if ( verbose )
   bu_log("Attempting to process region %s\n", db_path_to_string( pathp ));

ret_tree= process_boolean(curtree, tsp, pathp);

if ( ret_tree )
   r = ret_tree->tr_d.td_r;
else
{
   if ( verbose ) {
      bu_log( "\tNothing left of this region after Boolean evaluation\n" );
   }
   regions_written++; /* don't count as a failure */
   r = (struct nmgregion *)NULL;
}

regions_converted++;

if (r != (struct nmgregion *)NULL)
{
   struct shell *s;
   int empty_region=0;
   int empty_model=0;

   /* Kill cracks */
```

```
    s = BU_LIST_FIRST( shell, &r->s_hd );
    while ( BU_LIST_NOT_HEAD( &s->l, &r->s_hd ) )
    {
        struct shell *next_s;

        next_s = BU_LIST_PNEXT( shell, &s->l );
        if ( nmg_kill_cracks( s ) )
        {
      if ( nmg_ks( s ) )
      {
          empty_region = 1;
          break;
      }
        }
        s = next_s;
    }

    /* kill zero length edgeuses */
    if ( !empty_region )
    {
        empty_model = nmg_kill_zero_length_edgeuses( *tsp->ts_m );
    }

    if ( !empty_region && !empty_model )
    {
        process_triangulation(r, pathp, tsp);

        regions_written++;
    }

    if ( !empty_model )
        nmg_kr( r );
}

/*
 *  Dispose of original tree, so that all associated dynamic
 *  memory is released now, not at the end of all regions.
 *  A return of TREE_NULL from this routine signals an error,
 *  and there is no point to adding _another_ message to our output,
```

```
     *  so we need to cons up an OP_NOP node to return.
     */
    db_free_tree(curtree, &rt_uniresource);		/* Does an nmg_kr() */

    BU_GETUNION(curtree, tree);
    RT_TREE_INIT(curtree);
    curtree->tr_op = OP_NOP;
    return curtree;
}

/*
 * Local Variables:
 * mode: C
 * tab-width: 8
 * indent-tabs-mode: t
 * c-file-style: "stroustrup"
 * End:
 * ex: shiftwidth=4 tabstop=8
 */
```

ДОДАТОК В

Приклад формування NURBS-сплайну

```
/*                N U R B _ E X A M P L E . C
 * BRL-CAD
 *
/** @addtogroup nurb */
/** @{ */
/** @file librt/nurb_example.c
 *
 */
/** @} */

#include "common.h"

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "vmath.h"
#include "raytrace.h"
#include "nurb.h"
#include "plot2.h"

extern double drand48(void);

fastf_t grid[10][10][3];
fastf_t ngrid[10][10][3];


/* Interpoate the data using b-splines */
void
interpolate_data()
{
   struct face_g_snurb srf;
   struct face_g_snurb *srf2, *srf3;
   struct knot_vector new_kv;

   rt_nurb_sinterp(&srf, 4, (const fastf_t *)grid, 10, 10);
```

```
    /* lets take a look at it.  Refine to 100 points in both directions. */
    rt_nurb_kvknot(&new_kv, srf.order[0], 0.0, 1.0, 100, (struct resource *)NULL);
      srf2 = (struct face_g_snurb *) rt_nurb_s_refine(&srf, 0, &new_kv, (struct re-
source *)NULL);
    srf3 = (struct face_g_snurb *) rt_nurb_s_refine(srf2, 1, &new_kv, (struct resource
*)NULL);

    /* Draw refined mesh in yellow */
    pl_color(stdout, 200, 200, 50);
    rt_nurb_plot_snurb(stdout, srf3);
}

int
main(int argc, char *argv[])
{

    fastf_t hscale;
    fastf_t v;
    int i, j;

    if (argc > 1)
        printf("Usage: %s\n", argv[0]);

    hscale = 2.5;

    /* generate a 10 x 10 grid of random height data */

    pl_color(stdout, 155, 55, 55);

    for (i = 0; i < 10; i++) {
        for (j = 0; j < 10; j++) {
            v = hscale * drand48() + 10.0;
            grid[i][j][0] = i;
            grid[i][j][1] = j;
            grid[i][j][2] = v;

            pd_3move(stdout,
              grid[i][j][0],  grid[i][j][1], grid[i][j][2] - .14);
            pd_3cont(stdout,
```

```
            grid[i][j][0], grid[i][j][1], grid[i][j][2] + .14);

          pd_3move(stdout,
            grid[i][j][0] - .14, grid[i][j][1], grid[i][j][2]);
          pd_3cont(stdout,
            grid[i][j][0] + .14, grid[i][j][1], grid[i][j][2]);

          pd_3move(stdout,
            grid[i][j][0], grid[i][j][1]-.14, grid[i][j][2]);
          pd_3cont(stdout,
            grid[i][j][0], grid[i][j][1]+.14, grid[i][j][2]);
        }
    }

    interpolate_data();

    return 0;
}

/*
 * Local Variables:
 * mode: C
 * tab-width: 8
 * indent-tabs-mode: t
 * c-file-style: "stroustrup"
 * End:
 * ex: shiftwidth=4 tabstop=8
 */
```