bachelor's thesis

Application for design and export of racing tracks

Mykola Isaiev

May 21, 2020

supervisor: Ing. Ladislav Čmolík, Ph.D.

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Computer Graphics And Interaction

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Isaiev** Jméno: **Mykola** Osobní číslo: **466358**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Nástroj pro návrh a export závodních tratí

Název bakalářské práce anglicky:

Application for design and export of racing tracks

Pokyny pro vypracování:

Seznamte se s parametrickými polynomiálními křivkami a s metodami pro procedurální modelování silniční sítě. Na základě analýzy navrhněte a implementujte nástroj, který umožní navrhnout a exportovat 3D polygonální model závodní trati. Nástroj bude umožňovat definovat tvar závodní trati pomocí otevřené či uzavřené křivky a tvar profilu trati (příčný řez tratí) jako polygon. 3D model závodní trati vznikne tažením profilu trati po křivce definující tvar trati. Nástroj bude umožňovat uložit a znovu načíst tvar trati, tvar profilu a všechny ostatní vlastnosti závodní trati. Výsledkem exportu bude 3D polygonální model trati s texturovacími souřadnicemi, materiály jednotlivých částí trati (silnice, krajnice, chodník, trávník, apod.) a 2D textury. Exportovaný 3D model bude možné načíst do nástroje Cruden Panthera. Výslednou implementaci otestuje vytvořením alespoň pěti závodních tratí různých tvarů, jejich exportem a zobrazením v nástroji Cruden Panthera.

Seznam doporučené literatury:

Beneš B., J. Sochor, P. Felkel, J. Žára. Moderní počítačová grafika, 2. vydání. Computer Press, 2005.
Smelik R. M., T. Tutenel, R. Bidarra, and B. Beneš. A survey on procedural modelling for virtual worlds. Computer Graphics Forum, 33(6):31-50, 2014.
Galin E., A. Peytavie, N. Maréchal, and E. Guérin. Procedural generation of roads. Computer Graphics Forum, 29(2):429-438, 2010.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ladislav Čmolík, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2019** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Ladislav Čmolík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání	Podpis studenta

Acknowledgement

I would like to express my gratitude to Ing. Ladislav Čmolík, Ph.D. for the thesis supervision and countless thoughtful advice along the way. I would also like to thank my family and friends for the much needed moral support.

Declaration

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accordance with Methodical instructions about ethical principles for writing academic thesis.

Abstrakt

Vytváření trojrozměrných scén závodních tratí vyžaduje znalost specifického softwaru a může být časově náročné. Výzkumné centrum Smart Driving Solutions používá řidičský simulátor Cruden Panthera pro řadu interních a externích účelů. Simulátor poskytuje možnost používat vlastní scény závodních tratí, což je nezbytné pro výzkum. Tato bakalářská práce popisuje návrh a implementaci nástroje pro návrh a editaci závodních tratí, který by umožnil vytváření scén tratí a jejich následný import do Cruden Panthera.

Hlavní funkční požadavky aplikace byly analyzovány především z hlediska použitelnosti a integrace s Cruden Panthera. Závodní trať byla definována Bézierovou křivkou a mnohoúhelníkem profilu drahy. Byl prozkoumán proces importu scény do Cruden Panthera. Výsledky analýzy pak byly použity při návrhu a implementaci aplikace. Finální implementace editoru byla poté použita k vygenerování několika příkladů scén tratí a byl stanoven obecný směr pro budoucí vylepšení aplikace.

Klíčová slova

Závodné tratě, parametrické křivky, Bézierové křivky, paralelní křivky, Cruden Panthera, ASE formát, Qt

Abstract

The creation of auto race track three-dimensional scenes requires knowledge of specific software and can be time-consuming. The Smart Driving Solutions research center uses Cruden Panthera driving simulator for a number of internal and external purposes. The simulator provides the ability to use custom track scenes, which is necessary for the research. This thesis describes the development and implementation of a race track editor application that handles creation of track scenes and the consequential import into Cruden Panthera.

The main functional requirements of the application were analyzed from different standpoints. The emphasis during the analysis was put on usability and Panthera integration. An auto race track was defined with a Bézier spline and a profile polygon. Cruden Panthera track scene import process was studied. The analysis results were then used in the application design and during the implementation. The working implementation of the editor was then used to generate several track scene examples; and the general direction for future application improvement was set.

Keywords

Auto race tracks, parametric curves, splines, Bézier splines, offset curves, Cruden Panthera, ASE format, Qt

Contents

1. Introduction **1**
1.1. Project Aims . . . 1

2. Analysis of the task **2**
2.1. Analysis of ASE format and integration with Cruden Panthera . . . 2
2.1.1. ASE format review . . . 3
2.1.2. Configuration files for track scenes . . . 6
2.2. Race track geometry representation . . . 6
2.2.1. Race track layout as a set of turns . . . 7
2.2.2. Parametric curves and splines . . . 8
Catmull-Rom splines . . . 9
Bézier splines . . . 10
2.3. Brief introduction into Bézier splines . . . 11
2.3.1. De Casteljau's algorithm . . . 12
2.3.2. Ensuring continuity of a Bézier spline . . . 13
2.3.3. Polygonal approximation of Bézier splines . . . 14
2.4. Track profile . . . 15
2.5. The issue of curve offset . . . 16
2.5.1. Control polygon modification algorithms . . . 16
Tiller and Hanson algorithm . . . 17
Selective subdivision algorithm . . . 17
2.5.2. Polygonal approximation offset approach . . . 18
Naive point-wise offset . . . 19
Advanced point-wise offset . . . 19
2.6. Conversion from a parametric curve to polygonal mesh . . . 21
2.6.1. Texture coordinates and material data . . . 21
2.7. Analysis conclusion . . . 23

3. Race Track Editor application design **24**
3.1. Design requirements . . . 24
3.1.1. Usage goals . . . 25
3.1.2. Additional requirements . . . 25
3.2. Graphical user interface design . . . 26
3.3. Cruden Panthera import pipeline integration . . . 28
3.4. Design conclusion . . . 28

4. Implementation of Race Track Editor application **29**
4.1. Technologies used . . . 29
4.2. The application structure . . . 30
4.3. Output formats . . . 33

5. Results **35**
5.1. Known issues . . . 36

6. Conclusion **40**
6.1. Future work . . . 40

Appendices

A. CD contents **42**

Bibliography **43**

1. Introduction

This project has emerged as a collaboration with the Smart Driving Solutions research center. The research group works on developing advanced control systems for vehicles. Besides implementing the systems in real-life cars, a big part of the development process is computer simulations.

The purpose of a simulation is to provide an approximation of how a vehicle behaves under human control. The way simulation is executed may vary, but it generally includes running a driving simulator software, using a monitor to provide the driver with visual and audio information, and a steering wheel control for driver input.

The simulations have two major applications. They can be used for demonstration purposes providing images, videos, or interactive demos to back up the research publications. They also serve the internal purposes of the research as a testing environment used both for fine-tuning the control systems and collecting data during driving tests.

The research center uses several driving simulators which differ in their precision and flexibility. One of them is Cruden Panthera. Panthera is a package of software that includes a 3D graphics engine, its own physics engine, and several interfaces to connect custom physics controls as well as importing user-made 3D models and scenes.

The simulator has one default race track scene and does not provide any way to customize it, which is not enough. The research necessitates having a more diverse set of scenarios to run tests on. The solution is to create new tracks based on the research needs, but the process of creating a track scene requires knowledge of 3D editor software and can be time-consuming.

The aim of this thesis is to simplify the track creation process with a simple software that would replace 3D editors in the pipeline and make import in Cruden Panthera as simple as possible. To make the software more comfortable to use the user input is reduced into two dimensions only.

1.1. Project Aims

Over the course of this project several issues will be tackled. The first one is the specifics of Cruden Panthera track scene import. The software and file formats used to import race tracks will be analyzed. The next problem is the 2D representation of a 3D track scene. Different suitable geometric representations will be compared and improved upon. After that Race Track Editor application can be implemented. The implementation must meet the following functional requirements:

1. Ability to create and edit race track layout.
2. Ability to edit the track profile (cross-section).
3. Ability to export the track scene as a set of geometry mesh objects into a format used in Cruden Panthera.
4. Options to save and open track shape and profile along with other parameters.

On top of the functional requirements the application must be easy to learn and to use.

2. Analysis of the task

Before implementing the application the main functional requirements must be analyzed. Cruden Panthera's track scene import process can have a significant impact on the application architecture and must be studied first in order to provide a perspective for further analysis.

The next issue is of generating three-dimensional race track geometry based on the 2D input. It includes the description of existing standards for auto race tracks, the assessment of different geometrical definitions of a track from the standpoint of usability and Cruden Panthera integration, and the analysis of the mathematics required to transform two-dimensional user input into 3D scenes.

Over the course of this chapter the issues will be analyzed. The chapter also contains descriptions of algorithms and approaches used to solve some of the issues in the implementation.

2.1. Analysis of ASE format and integration with Cruden Panthera

One of the most important requirements for the developed application is to be compatible with Cruden Panthera driving simulator. The application must fit the track scene import pipeline which can influence the implementation substantially. It sets the formatting rules for the output. It also dictates what information can and cannot be imported into Panthera and sets limitations to the achievable level of detail.

Unfortunately the available documentation for the software is rather minimal and a large part of the specifications had to be either reverse-engineered from existing examples of working track scenes or figured out by trial and error. As it was briefly mentioned in the introduction Cruden Panthera includes several stand-alone applications used for different aspects of the simulation. The application for importing track scenes is called TrackEd.

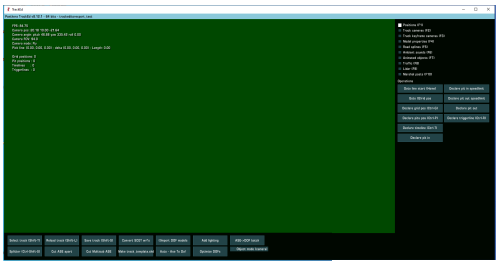


Fig. 1. TrackEd application default window with no track scene loaded

Cruden Panthera tends to be very strict about the location of the files used in simulations. All information about track scenes and car objects must be located in the dedicated path. The file structure inside the folders also plays an important role as all the parameters are stored in files with special naming conventions. TrackEd only has access to the track scene sub-folders located in the tracks folder. In order to be able

to launch a simulation on a track its folder must contain the individual files for track information specifications, geometry objects list, geometry sources, information about start line and finish line as well as pit-stop positions, and shader descriptions.

Besides importing track scenes into the internal format, TrackEd application allows to preview a track scene with a flying camera. It also provides functions to set property flags for objects, to add ambient sounds, to create predefined camera positions and movements, and other. The property flags include collision flags and material specifications such as the ability to receive shadows. The functionality for any further modification of models themselves is not present.

The only way to add models to the track scene is to import them from a file. This is done in two steps: the input file has to be converted into geometry source files, then the source files have to be loaded into the scene. The finishing touch is to save the track scene, which generates the list of geometry objects in the scene and saves other specifications into the corresponding files. This approach to save track data into segregated files allows to easily copy configurations from one track scene into another.

2.1.1. ASE format review

In order to import a model into TrackEd it must be stored as a set of vertices and faces. In computer graphics 3D objects are often stored as a set of interconnected points in space. A vector of all information about a point is often referred to as vertex. It usually contains tuples of integers and real numbers. The role of each number is determined by its position in the vector. Besides the coordinates in space, a vertex can include individual color information, texture coordinates, information about adjacent edges and faces, and other.

Some information about vertices requires a different segmentation of data to be stored properly (e.g. each vertex may have a separate normal vector for each polygon it is included in). For those cases vertices are often grouped in triplets called faces. A couple of faces sharing an edge is called a quad. A single point position can be included in multiple different faces having different additional data. A set of faces and edges containing all vertices is commonly called geometry mesh.

TrackEd requires the input file to be in ASCII Scene Export format (often referred to by the associated file extension ASE). This file format is native to Autodesk 3D studio Max software and is supported in the relevant versions (3ds MAX 2019) mostly due to legacy reasons.

As the name of the format suggests it stores information about a 3D scene as readable text. The format has a hierarchical data organization aimed mostly to enhance readability. Nowadays ASE files are used rarely. This is due to their extreme inefficiency and emphasis on readability. For modern purposes binary formats or text alternatives with more robust data organizations are commonly selected. Due to how archaic and rarely used the format is, the specifications accessible to public are very limited [1]. The following description is based in part on the experience of working with the format in context of Panthera integration.

An ASE file contains several sections of data denoted by corresponding tags starting with an asterisk (e.g. "*SCENE") followed by an opening curly bracket. Closing curly bracket constitutes the end of a section. Most data sections can have subsections and data properties. A subsection behaves in the same way as a section. Data property is a tag starting with an asterisk and followed by a number or a sequence of numbers sometimes divided by letters (e.g. "*MESH_FACE 0: A: 2 B: 0 C: 3"). Following is an example of a simple 3D scene exported into ASE format (listing 2.1).

Listing 2.1 The content of an ASE file describing a scene with a single quad in it

```
*3DSMAX_ASCIIEXPORT 200
*COMMENT "AsciiExport Version  2.00 - Mon Apr  6 18:47:59 2020"
*SCENE {
  *SCENE_FILENAME ""
  *SCENE_FIRSTFRAME 0
  *SCENE_LASTFRAME 100
  *SCENE_FRAMESPEED 30
  *SCENE_TICKSPERFRAME 160
  *SCENE_BACKGROUND_STATIC 0.0000 0.0000  0.0000
  *SCENE_AMBIENT_STATIC 0.0000  0.0000  0.0000
}
*MATERIAL_LIST {
  *MATERIAL_COUNT 0
}
*GEOMOBJECT {
  *NODE_NAME "Plane001"
  *NODE_TM {
    *NODE_NAME "Plane001"
    *INHERIT_POS 0 0 0
    *INHERIT_ROT 0 0 0
    *INHERIT_SCL 0 0 0
    *TM_ROW0 1.0000 0.0000  0.0000
    *TM_ROW1 0.0000 1.0000  0.0000
    *TM_ROW2 0.0000 0.0000  1.0000
    *TM_ROW3 0.0000 0.0000  0.0000
    *TM_POS 0.0000  0.0000  0.0000
    *TM_ROTAXIS 0.0000  0.0000  0.0000
    *TM_ROTANGLE 0.0000
    *TM_SCALE 1.0000  1.0000  1.0000
    *TM_SCALEAXIS 0.0000  0.0000  0.0000
    *TM_SCALEAXISANG 0.0000
  }
  *MESH {
    *TIMEVALUE 0
    *MESH_NUMVERTEX 4
    *MESH_NUMFACES 2
    *MESH_VERTEX_LIST {
      *MESH_VERTEX    0 -20.0000  -10.0000  0.0000
      *MESH_VERTEX    1 20.0000 -10.0000  0.0000
      *MESH_VERTEX    2 -20.0000  10.0000 0.0000
      *MESH_VERTEX    3 20.0000 10.0000 0.0000
    }
    *MESH_FACE_LIST {
      *MESH_FACE    0:    A:    2 B:    0 C:    3 AB:    1 BC:    0 CA:    1  *MESH_SMOOTHING 1 *MESH_MTLID 0
      *MESH_FACE    1:    A:    1 B:    3 C:    0 AB:    1 BC:    0 CA:    1  *MESH_SMOOTHING 1 *MESH_MTLID 0
    }
    *MESH_NUMTVERTEX 8
    *MESH_TVERTLIST {
      *MESH_TVERT 0 0.0000  0.0000  0.0000
      *MESH_TVERT 1 1.0000  0.0000  0.0000
      *MESH_TVERT 2 0.0000  0.0000  0.0000
      *MESH_TVERT 3 1.0000  0.0000  0.0000
      *MESH_TVERT 4 0.0000  0.0000  0.0000
```

```
54 ⎵⎵⎵⎵⎵*MESH_TVERT 5⎵1.0000⎵⎵0.0000⎵⎵0.0000
55 ⎵⎵⎵⎵⎵*MESH_TVERT 6⎵0.0000⎵⎵1.0000⎵⎵0.0000
56 ⎵⎵⎵⎵⎵*MESH_TVERT 7⎵1.0000⎵⎵1.0000⎵⎵0.0000
57 ⎵⎵⎵⎵}
58 ⎵⎵⎵⎵*MESH_NUMTVFACES 2
59 ⎵⎵⎵⎵*MESH_TFACELIST {
60 ⎵⎵⎵⎵⎵*MESH_TFACE 0⎵6⎵4⎵7
61 ⎵⎵⎵⎵⎵*MESH_TFACE 1⎵5⎵7⎵4
62 ⎵⎵⎵⎵}
63 ⎵⎵⎵⎵*MESH_NUMCVERTEX 0
64 ⎵⎵⎵⎵*MESH_NORMALS {
65 ⎵⎵⎵⎵⎵*MESH_FACENORMAL 0⎵⎵0.0000⎵⎵0.0000⎵⎵1.0000
66 ⎵⎵⎵⎵⎵⎵⎵*MESH_VERTEXNORMAL 2⎵⎵0.0000⎵⎵0.0000⎵⎵1.0000
67 ⎵⎵⎵⎵⎵⎵⎵*MESH_VERTEXNORMAL 0⎵⎵0.0000⎵⎵0.0000⎵⎵1.0000
68 ⎵⎵⎵⎵⎵⎵⎵*MESH_VERTEXNORMAL 3⎵⎵0.0000⎵⎵0.0000⎵⎵1.0000
69 ⎵⎵⎵⎵⎵*MESH_FACENORMAL 1⎵⎵0.0000⎵⎵0.0000⎵⎵1.0000
70 ⎵⎵⎵⎵⎵⎵⎵*MESH_VERTEXNORMAL 1⎵⎵0.0000⎵⎵0.0000⎵⎵1.0000
71 ⎵⎵⎵⎵⎵⎵⎵*MESH_VERTEXNORMAL 3⎵⎵0.0000⎵⎵0.0000⎵⎵1.0000
72 ⎵⎵⎵⎵⎵⎵⎵*MESH_VERTEXNORMAL 0⎵⎵0.0000⎵⎵0.0000⎵⎵1.0000
73 ⎵⎵⎵⎵}
74 ⎵⎵}
75 ⎵⎵*PROP_MOTIONBLUR 0
76 ⎵⎵*PROP_CASTSHADOW 1
77 ⎵⎵*PROP_RECVSHADOW 1
78 ⎵⎵*WIREFRAME_COLOR 0.3412⎵0.8824⎵⎵0.3412
79 }
```

The main sections feature two leading headers, which bare no information about the scene itself, general scene information, the list of all materials in the scene, and one or more geometry object sections. Each material can contain a subsection called "*SUBMATERIAL". TrackEd requires all the materials in a track scene to be a part of one material, so the scene must contain a single material with as many sub-materials as needed.

Geometry object section includes the name of the object, its transformation matrix, and Mesh subsection. The subsection contains the list of vertices and faces both in spacial and texture coordinates, and the list of face and vertex normal vectors. Tag "*MESH_SMOOTHING" corresponds to 3Ds Max smoothing groups. A material can be assigned to an object using tag "*MATERIAL_REF" followed by the material ID. Tag "*MESH_MTLID" refers to the index of a sub-material in the assigned material. Texture coordinates allow for 3 dimensions, however the third dimension is used very rarely. The texture coordinates are not restricted to be between 0 and 1. The native 3Ds Max exporter creates several redundant texture vertices, but the scene can be imported into Panthera without them.

ASE format uses inconsistent spacing convention. It combines tabs and regular spaces. Panthera relies on preserving the space types for every parameter. Breaking the convention will not lead to an error on import, but may cause undefined behaviour for some parameters.

Cruden Panthera only uses a part of scene information accessible in an ASE file. It uses its own shader definition system and the material data inside input files is mostly discarded. It also relies on the normal vectors predefined in the input file and does not provide any way to modify them inside TrackEd. Panthera does not have explicit restrictions for the maximum number of objects or faces in a scene, however, there is a limit for the number of faces per material. This means that if the limit for a material

is surpassed, a new material identical to the original must be created and assigned to a part of the geometry.

2.1.2. Configuration files for track scenes

As it was mentioned earlier, Panthera track configurations are stored in separate files. The files are readable and have identical syntax. Some of them can be modified from within TrackEd, others have to be edited manually. Following is the list of configuration files required for Panthera to recognize the track folder as a track.

- **track.ini** – this file contains general information about a track, which includes the name and the unique identifier of the track. It has to be created by hand.
- **special.ini** – this file describes the position of the start and finish lines, pit-stop positions, some information for the internal physics engine, and information about scene lighting. Parts of it can be modified within TrackEd.
- **geometry.ini** – it contains the list of all geometry objects in the scene with links to the corresponding source files and flag information. It is created and updated automatically, whenever the track in TrackEd is saved.
- **track.shd** – it includes shaders definition for the entire scene. Shader names must start with "shader_" followed by the name of the corresponding sub-material in the input file. Every shader must derive (denoted by "∼" sign) from one of the predefined basic shaders. A standard shader has one texture map for diffuse color, thought the system allows creating more complicated shaders with normal maps and transparency. The file must be modified manually.

Besides the configuration files geometry source files must be provided in the paths listed in geometry.ini.

A significant part of the configuration data does not have to be generated for every new track. The research group does not use Panthera's internal measuring tools, because of that the content of special.ini can be copied from an exisiting track scene folder. The same set of shader definitions can also be used by multiple tracks. For this reason a package of necessary configuration files will be provided along with the ASE output of the application for users to copy into the desired folder.

2.2. Race track geometry representation

Another important requirement for the developed application is to provide a way to create and modify race track scenes. A real-life auto race track has a very complex set of parameters which varies drastically for different disciplines. The restrictions for the length and width of a track, its surface and layout are unique for every race type. Some tracks can be built on an existing location such as a part of a city or off-road landscape. Others must be created from scratch.

Despite all the variety many disciplines share the main geometric characteristics of a track: each part has some length, width, curvature, altitude, and camber (a parameter that describes variations in altitude along the width of the track) [2]. Then a race track can be defined as a function that takes distance from the start along the middle line of the track and outputs all the parameters listed above.

The application must allow users to specify the track function. In order to reduce user input into 2 dimensions the function can be divided into two independent component functions: layout and profile. The layout component corresponds to curvature and length of each part of a track as well as the changes in overall altitude. In other words the layout component is the definition of the middle line of the track. The profile component includes width and camber of a track.

It must also be noted that the research group from Smart Driving Solutions is currently on the beginning stages of their development and they do not test on any tracks with variable height. Even though addition of the third dimension is something to consider in the future, in this project the attention will be focused on creating mostly flat tracks. This makes both component functions of the track strictly two-dimensional.

Another advantage of such function decomposition has to do with the visual representation of a track. It can be beneficial to define road markings in a similar manner, because they often follow the same pattern of stretching a predefined set of marks along the road. It can also play out nicely when it comes to adding textures to the race track model.

There are no explicit limitations for an auto race track layout. Different disciplines limit the geometrical makeup of the layout differently. Formula One tracks are usually divided into straight parts and turns. A turn is described with angle, inner radius, and outer radius. However, this is not a strict rule and many tracks feature turns with inconsistent radii (e.g. Shanghai International Circuit - figure 2). Because of that, the 2D representation of a race track layout is not strictly formalized and can be selected based on flexibility and usability.

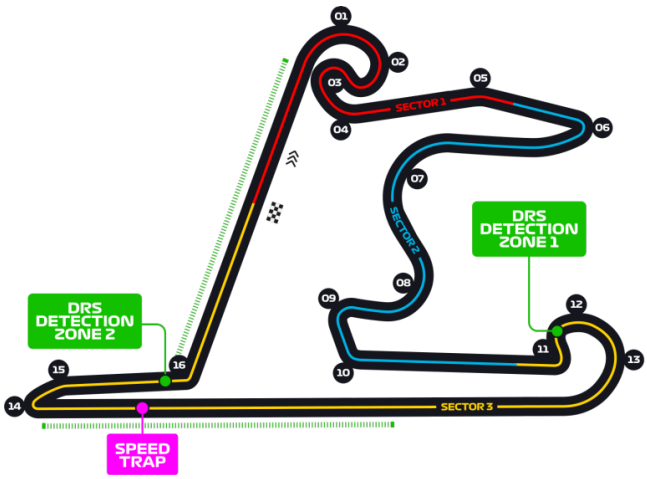


Fig. 2. Map of Shanghai International Circuit [3]

The following sections are dedicated to the analysis of different ways to define the layout function. The main focus of the analysis is the formalization usability.

2.2.1. Race track layout as a set of turns

The intuitive approach to represent a race track as a set of turns connected by straight lines (figure 3) has its advantages and disadvantages.

The main advantage of this representation is that it coincides with some real-life auto race track norms. It also involves less complicated mathematics comparing to the alternatives listed below. It allows for a more rational distribution of details and simplifies conversion of a thin line into a race track.

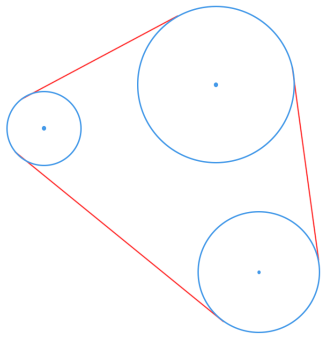


Fig. 3. A race track as a set of circles and tangent lines

The main disadvantage of the representation is how restrictive it is. As it was stated earlier, a race track does not have to fit this formalization. The representation might not be very user friendly. It adds a layer of complications to creation of more fluid shapes. This approach is not very common in computer graphics and does not have a strong mathematical foundation as the alternatives do.

Because of its disadvantages this approach to race track geometry representation will not be used in this project.

2.2.2. Parametric curves and splines

A good candidate for track layout definition is a parametric curve. These curves are widely used in computer graphics for accomplishing a huge spectrum of goals. Among other applications they can serve as a controller providing a user with an in-depth grasp of the behaviour of some dynamically changing parameters - the curves are the world standard tool for animation - as well as they are one of the most useful 3D modeling tools.

A general **parametric curve** C is defined as follows [4]:

$$C : \mathbb{R} \to \mathbb{R}^n, f_i : \mathbb{R} \to \mathbb{R}, i = 1...n;$$
$$C(t) = (f_1(t), f_2(t), ..., f_n(t))$$

The parametric curve is polynomial, if all component functions f_i are polynomial functions. The degree of a polynomial parametric curve is determined by the highest degree of the functions f_i.

A **parametric spline** S is a piecewise parametric curve [5] which means the following:

$$S : [a, b] \to \mathbb{R}^n,\ a, b \in \mathbb{R},\ t_i \in [a, b],\ i = 0...m,$$
$$a = t_0 \leq t_1 \leq ... \leq t_m = b,\ C_i : \mathbb{R} \to \mathbb{R}^n;$$
$$S(t) = \begin{cases} C_0(t) & \text{if } t \in [t_0, t_1] \\ C_1(t) & \text{if } t \in [t_1, t_2] \\ ... & \\ C_{m-1}(t) & \text{if } t \in [t_{m-1}, t_m] \end{cases}$$

Here vector $\mathbf{t} = (t_0, ..., t_m)$ is called a **knot vector**. If $\forall j, k = 1...m : t_j - t_{j-1} = t_k - t_{k-1}$, the spline is **uniform**. A **curve segment** is a spline whose knot vector contains only two elements. Any spline is constructed by chaining together one or more

curve segments so that the end of a previous segment coincides with the beginning of the next one. A parametric spline $S : [a, b] \to \mathbb{R}^n$ is **closed**, if $S(a) = S(b)$.

A spline S is of **continuity** C^n, if it has continuous n-th derivation in all points.

Natural cubic splines are a family of cubic polynomial parametric splines that contains both Catmull-Rom splines and cubic Bézier splines.

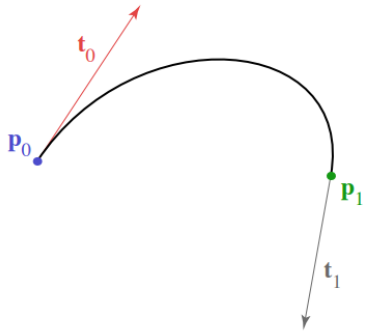


Fig. 4. A Hermite spline (a type of natural cubic spline) [6]

In case of two dimensions a natural cubic spline H is defined the following way [7]:

$$H : [0, 1] \to \mathbb{R}^2,\ \alpha \in [0, 1],\ V_{i-1} \in \mathbb{R}^2,\ M \in \mathbb{R}^{4\times 4},\ i = 0, ..., 3;$$

$$H(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} M \begin{bmatrix} V_{-1} \\ V_0 \\ V_1 \\ V_2 \end{bmatrix}$$

Here V_{i-1} are control points of the curve. Matrix M influences the behavior of the spline between the control points. **Control polygon** of a polynomial parametric spline is the set of all its control points.

The matrix M significantly influences the behaviour of the spline, but it cannot increase the degree of the function or change the function type to be anything else than a polynom. Because of that there exist ways to convert between the natural spline types by applying trivial control polygon modifications. For this reason a general natural cubic spline is often referred to as a basis spline. However, different types of splines are usually tailored for different purposes and can have varying usability in a given scenario.

Catmull-Rom splines

A Catmull-Rom spline P is a natural cubic spline with a cardinal matrix and tension parameter 0.5 [8]:

$$P(t) = \frac{1}{2} \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_{-1} \\ V_0 \\ V_1 \\ V_2 \end{bmatrix}$$

A Catmull-Rom spline passes through every control point (figure 5) and is not contained within their convex hull. Due to its continuity degree (the spline has C^1 continuity which is just enough for generating a road-like geometry) Catmull-Rom can be a

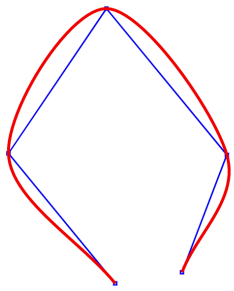


Fig. 5. A Catmull-Rom spline with 4 segments [9]

good candidate for the task. Another advantage of this type of splines is their efficiency for the calculations.

A Catmull-Rom spline is constructed using 4 control points, however a single spline can have any number of control points greater than 4. This is achieved by combining multiple cubic spline segments each of which uses a subset of 4 neighboring control points from the original set of points. To construct a spline with 2 segments it would take 5 control points, 6 for 3 segments and so on.

This type of splines is widely used as a way to approximate interpolation between a set of points. They are not as widely used as a modeling tool in computer graphics. They provide very limited control over the shape of the curve in between the points. Because of this they will not be used in this project.

Bézier splines

A cubic Bézier spline B is a natural cubic spline whose matrix M is Bernstein matrix [10]:

$$B(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_{-1} \\ V_0 \\ V_1 \\ V_2 \end{bmatrix}$$

Unlike Catmull-Rom splines a Bézier spline can be of any positive whole degree. To construct a Bézier curve segment of n-th degree $n+1$ control points are required. The segment passes through the first and the last point of its control polygon. The rest of the control points are used to influence the behaviour of the curve. Quadratic and cubic Béziers are the most common in computer graphics due to the efficiency of computation and high usability.

Bézier splines are a common modelling tool. They allow to control the curve in-between the knot points and can describe a large portion of natural splines including Catmull-Rom splines. They fit inside their control polygon convex hull and form a natural arc-like shape between two handles. They are relatively easy to manipulate with a pointing device like a computer mouse and flexible even in a beginner's hands.

One of the biggest disadvantages of Bézier splines is that approximating a circular arc is not exactly straight-forward. A quadratic Bézier segment can only effectively approximate a small portion of a circle. Cubic splines are better fit for the task, but the approximation can be difficult to set up by hand.

Despite its disadvantages Bézier splines are selected as the representation of race track layout in this project because of their high usability. The following section is dedicated to furthering the mathematical foundation of Bézier splines.

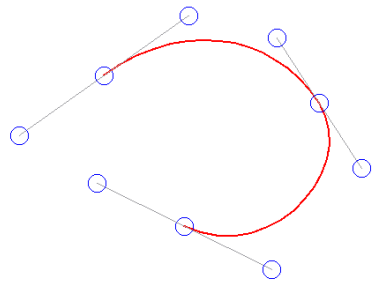


Fig. 6. A cubic Bézier spline with 2 segments

2.3. Brief introduction into Bézier splines

As it was stated earlier general Bézier splines exceed the family of natural cubic splines (figure 7).

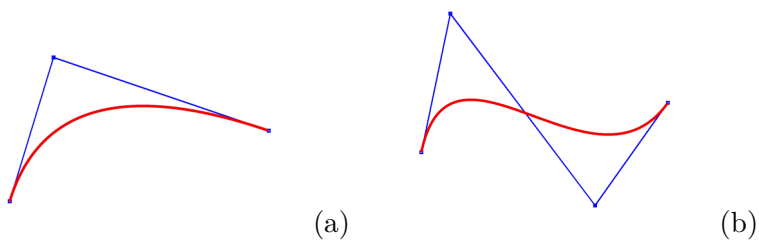


Fig. 7. Quadratic (a) and cubic (b) Bézier splines [9]

The definition of a general Bézier segment relies on **Bernstein polynomials**. Bernstein polynomial B of n-th degree is defined as follows [11]:

$$B^n : [0,1] \to \mathbb{R}^n,\ B^n(u) = (B_0^n(u), ..., B_n^n(u));$$
$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i},\ i = 0, ..., n.$$

A general Bézier spline S of n-th degree can now be written as following:

$$S : [0,1] \to \mathbb{R}^2, V_j \in \mathbb{R}^2, j = 0, ..., n;$$
$$S(t) = \sum_{i=0}^{n} B_i^n(t) V_i.$$

For a spline with 4 control points (degree $n = 3$) this notation is equivalent to a natural cubic spline with Bernstein matrix. For a quadratic Bézier segment S the matrix notation has the following form [10]:

$$S : [a,b] \to \mathbb{R}^2, V_j \in \mathbb{R}^2, a, b \in \mathbb{R}, j = -1, 0, 1;$$
$$S(t) = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_{-1} \\ V_0 \\ V_1 \end{bmatrix}.$$

2.3.1. De Casteljau's algorithm

The geometrical implication of Bernstein polynomials is known as **De Casteljau's algorithm** for spline evaluation. The main idea of the algorithm is to find a point on the curve by iteratively applying linear interpolation to the pairs of neighboring points in its control polygon.

The input of the algorithm is a Bézier curve segment with knot vector $\{a, b\}$ and a parameter $t \in [a, b]$. The algorithm's output is a point on the curve segment and new control polygon containing twice as many control points. The new polygon defines two new Bézier curve segments of the same degree as the input curve. The shape of the spline constructed from the new segments matches the original curve exactly.

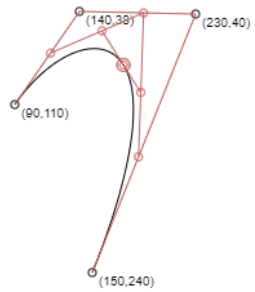


Fig. 8. De Casteljau's algorithm applied to evaluate a cubic Bézier [12]

This operation of dividing a segment into two without changing the shape of the spline is often referred to as **split**.

The steps of De Casteljau's algorithm with input segment $P(t) : [0, 1] \to \mathbb{R}^2$ of n-th degree, input parameter $t \in [0, 1]$, and output segments $S_1, S_2 : [0, 1] \to \mathbb{R}^2$ of the same degree n are following [13]:

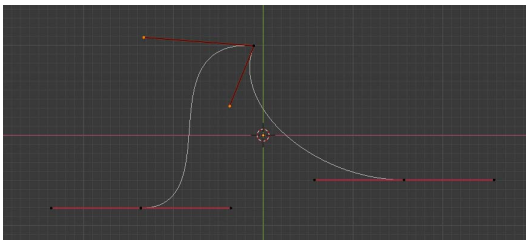
1. Let $V \in \mathbb{R}^{n\times 2}$ be the set of control points of the original curve $P(t)$.
2. Let $U \in \mathbb{R}^{m\times 2}$ be the set of control points of the first segment $S_1(t)$. At the start of the algorithm $m = 0$. Analogically $W \in \mathbb{R}^{k\times 2}$ is the control polygon for $S_2(t)$, $k = 0$.
3. Let $t \in (0, 1)$ be the split point.
4. If the number of points in V is one, add the point to U and W. Go to step 6.
5. Else:
   a) Add the first point in V to U;
   b) Add the last point in V to W;
   c) Let $V_1 \in \mathbb{R}^{l\times 2}$ be a new set of points such that $l = n - 2$;
      - For every point in V except for the first and the last one add linear interpolation of every unordered pair of neighboring points to the V_1 set such that:
      - $V_{1i} = tV_i + (1 - t)V_{i+1}$, $i = 1, ..., l$;
   d) Let V_1 be V. Go to the step 4.
6. The last point in U is lies on the curve. Terminate.

It should be noted that the knot vector of the result spline does not match the original knot vector.

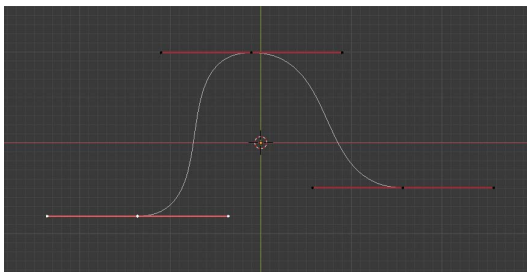
De Casteljau's algorithm is used widely for evaluating and editing Bézier splines. Applied recursively it can result in a more optimal sample distribution. This has a positive impact on the performance which is a big priority in computer graphics.

2.3.2. Ensuring continuity of a Bézier spline

It is natural to expect a race track layout to be described with a C^1 smooth curve. A Bézier segment has infinite continuity (for any $t \in (0, 1)$). However, combining multiple segments into a spline does not guarantee any continuity beyond C^0. In order to ensure higher degrees of continuity in the knot points of the spline additional restrictions must be applied. To guarantee continuity of the first derivation any abrupt changes in the derivation vector must be prevented.



(a)



(b)

Fig. 9. Cubic Bézier splines with C^0 (a) and C^1 (b) continuity (screenshot taken in Blender 2.8)

Because of the geometrical nature of the curves the first edge of every segment's control polygon coincides with its first derivation in its first knot ($S'(a)$). The last edge coincides with the vector opposite to the first derivation in the last knot ($-S'(b)$). Therefore a uniform Bézier spline is C^1 continuous, if for every segment of the spline the last point of its control polygon lies exactly in the middle between the previous point and the second point of the next segment's control polygon[14].

Another way to state it is: uniform Bézier spline S of n-th degree with knot vector $\mathbf{k} = \{k_0, ..., k_{m+1}\}$ and control polygon $\mathbf{P} = \{P_0, ..., P_{mn+1}\}$ is C^1 smooth, if

$$\forall j = 1, ..., m : P_{jn} - P_{jn+1} = P_{jn+2} - P_{jn+1}.$$

For nonuniform splines the restriction must be modified the following way:

$$\forall j = 1, ..., m : P_{jn} - P_{jn+1} = \frac{k_j - k_{j-1}}{k_{j+1} - k_j}(P_{jn+2} - P_{jn+1}).$$

This holds true if all knots are unique.

The new restriction on the splines control polygons has a special consequence of quadratic Bézier splines. Control polygon of a quadratic Bézier segment contains 3

points. All of them are effected by the restriction. This makes the splines much less usable, because changes in one segment yield changes in every segment of the spline. Due to this issue the project will only utilize cubic Bézier splines.

2.3.3. Polygonal approximation of Bézier splines

In computer graphics true Bézier splines are often reduced to a polygonal representation using a finite number of evaluations. This approach is natural to display a 2D curve on a screen with a limited resolution. It is also used in 3D graphics to increase computation efficiency and as a way to apply non affine transformations to an arbitrary Bézier curve.

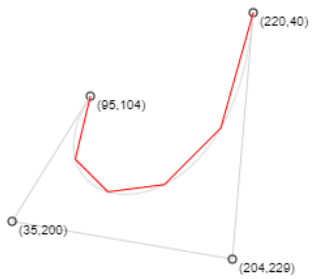


Fig. 10. Polygonal approximation of a cubic Bézier segment.[12]

A polygonal approximation $P \in \mathbb{R}^{n\times 2}$ of a Bézier segment $B : [0,1] \to \mathbb{R}^2$ can be defined as follows:

$$P = \{p_i | \exists t_i \in [0,1] : B(t_i) = p_i; i = 1, ..., n\},$$
$$\forall p_i \in P : t_i \leq t_{i+1}.$$

A polygonal approximation of a Bézier $B : [a,b] \to \mathbb{R}^2$ can be generated in a number of ways. The most straight-forward approach is to select a subdivision step d such that $(b-a)/d = n$ and evaluate the spline in points $a + dj, j = 0, ..., n$.

An alternative way is to use a **recursive version of De Casteljau's algorithm**. In each iteration of the algorithm the segment $B : [a,b] \to \mathbb{R}^2$ is split in point $t = 0.5(a+b)$. The control polygons of each new segment are then evaluated to check, whether a sufficient precision of approximation was reached. If the precision suffices the segment is represented with a single edge connecting the first and the last point in its control polygon. Else the algorithm is applied on the segment.

The steps of the algorithm are the following:

1. Let $B : [a,b] \to \mathbb{R}^2$ be a Bézier segment. Let $d \in \mathbb{R}$ be the approximation threshold.
2. Let $P \in \mathbb{R}^{n\times 2}, n \in \mathbf{N}$ be the result polygonal approximation.
3. Add the first point from the control polygon of B to the approximation P.
4. Split B in $t = 0.5(a+b)$ into two segments.
5. For each new segment B_i:
   a) If the control polygon area S_i is below the approximation threshold (i. e. $S_i \leq d$), add the last point of the control polygon to the approximation P.
   b) Else let $B = B_i$. Go to step 5.

6. P is the polygonal approximation of the Bézier segment B. Terminate.

The recursive De Casteljau's algorithm leads to a more rational evaluation spread and fits the curve more tightly comparing to the straight-forward approach. For this reason it will be used in the project as the method to approximate Bézier curves.

2.4. Track profile

Even though the project is mainly focused on creating flat tracks and changes in chamber will not be used by the researchers at the current stage of their development, a more sophisticated profile definition can be beneficial in a number of ways. The most important advantage has to do with the visuals of a track. A 3-dimensional profile can influence the appearance of a track without interfering with the physics model. It can be used to create convincing track borders. It also provides a way to define additional material and texture information even for a completely flat shape. Another benefit of having more flexible way to define profile is that it would make it easier to add the third dimension in the future.

Fig. 11. Example of a track profile (here gray vertical line is the position of the middle curve, circles represent individual vertices, and edge colors corresponds to different material presets)

Track profile can be defined as an open polygon. Each vertex of the polygon describes the position of some points of the track against its middle curve. It should also contain material data.

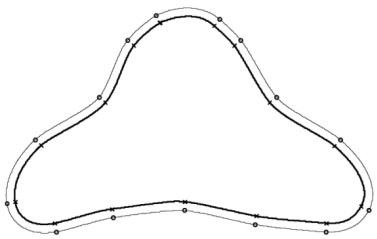


Fig. 12. Offset curve example [15]

Instead of defining the profile for each point of the track manually a single definition can be extrapolated on the entire track. This requires generating a number of curves parallel to the track layout curve (figure 12). Each new curve represents a vertex from the profile polygon and the area between neighboring curves corresponds to an edge of the polygon. The most straight-forward approach to generate the parallel curves is to apply offset to the layout curve. The next section is dedicated to the specifics of the offset curves generation process.

2.5. The issue of curve offset

Given a general two-dimensional parametric curve C, an **offset curve** $C_d : \mathbb{R} \to \mathbb{R}^2$ by radius d is defined as follows [16]:

$$C : \mathbb{R} \to \mathbb{R}^2,\ C(t) = (x(t), y(t));$$
$$C_d(t) = C(t) + dN(t).$$

Here $N(t)$ is the unit normal vector of $C(t)$ and is defined the following way:

$$N : \mathbb{R} \to \mathbb{R}^2;$$
$$N(t) = \frac{(y'(t), -x'(t))}{\sqrt{x'(t)^2 + y'(t)^2}}.$$

For any natural polynomial parametric curve C the offset curve C_d does not have to be a polynomial function. The group of polynomial curves for which an offset curve is also polynomial of the same degree is very limited and does not include Bézier splines nor any other general natural splines.

There exist multiple offset curve approximation algorithms that avoid evaluating true offset curves. The most common approach to calculate offset curve of a Bézier spline aims to preserve the original type and degree of the curve. It is based on manipulating the control polygon of the original Bézier spline to generate the offset approximation as another Bézier spline of the same degree.

An alternative way does not restrict the offset curve to be a polynomial spline. It is based on polygonal representation of the offset curve and numeric evaluations.

2.5.1. Control polygon modification algorithms

The control polygon of a Bézier segment can be modified by applying two operations. The segment $B : [a, b] \to \mathbb{R}^2$ can be split at an arbitrary point $t \in [a, b]$ creating a new control polygon. Alternatively the offset transformation can be applied to the polygon. The transformation can be formalized the following way:

$$N, P_0, P_1 \in \mathbb{R}^{n\times 2}, d \in \mathbb{R}^n,$$
$$P_{1i} = d_i(P_{0i} + N_i), i = 1, ..., n.$$

Here P_0 is the original control polygon, P_1 is the new polygon, d is the vector of offset distances, and N is a set of offset vectors for each control point.

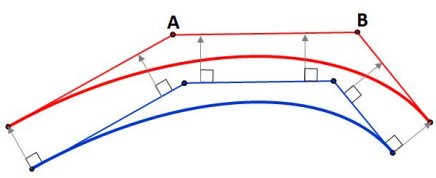


Fig. 13. Offset transformation [17]

The offset is then performed by applying a sequence of operations with different parameters t, d, and N. The offset distance does not have to be constant and can be adjusted to ensure a more accurate approximation.

Tiller and Hanson algorithm

Tiller and Hanson algorithm is a recursive control polygon modification algorithm. It is based on the fact that a single offset transformation produces better offset curve approximation for a flatter Bézier segment (i.e. its control polygon approaches a flat line connecting its first and last points).

The algorithm does not use any additional heuristics. The distance vector and offset vectors are selected in such way, that the offset control points coincide with the offset legs intersection points. The algorithm iterates over the entire spline, subdivides and offsets each segment until a sufficient approximation precision is reached.

The steps of the algorithm are following [18]:

1. Let C be the original curve and P the control polygon;
2. Offset each leg of P by offset distance;
3. Intersect offset legs to find new control points of the offset curve;
4. Calculate the deviation of the approximated offset curve;
   - If the deviation is greater than the threshold, divide the original C into two segments at $t = 0.5$;
   - For each new segment S_i let C be S_i and go back to the step 2;
   - Else terminate.

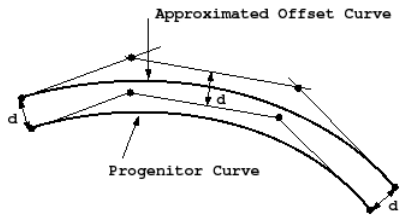


Fig. 14. Offset transformation in Tiller and Hanson algorithm [18]

This algorithm performs well on quadratic Bézier splines, but does not guarantee a consistent offset distance for cubic curves [16]. The algorithm also relies on recursion and can take more iterations to solve inconsistencies comparing to some alternatives.

Selective subdivision algorithm

Selective subdivision introduces a simple heuristic to Tiller and Hanson approach. Instead of dividing a segment $B : [0, 1] \to \mathbb{R}^2$ at a constant $t = 0.5$, the split point is selected based on the control polygon.

The heuristic is to select points of curve extremities as subdivision points. For a polynomial parametric curve $C : [a, b] \to \mathbb{R}^2, C(t) = (f_1(t), f_2(t))$ an extremity is a point $t \in [a, b]$ where a component function f_i has an extreme point.

In case of a quadratic Bézier segment the parameter t is selected in such way, that distance from $B(t)$ to the second point in the control polygon is the smallest.

The steps of the algorithm are following [19]:

1. Let $B : [a, b] \to \mathbb{R}^2$ be a quadratic Bézier segment with control polygon $P \in \mathbb{R}^{3\times 2}$.
2. Let $k \in \mathbf{N}$ be a given number of subdivisions.

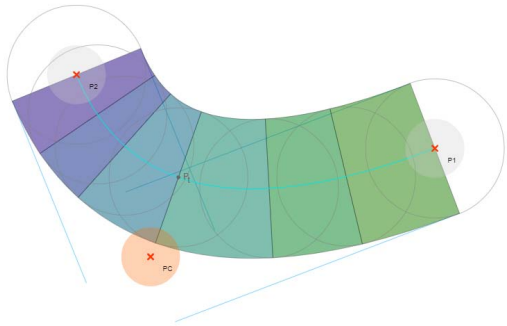


Fig. 15. Selective subdivision offset of a quadratic spline [19]

3. Let $A : [a, b] \to \mathbb{R}^2$ be the final approximation spline.
4. Find an extremity $t \in [a, b]$. Split B in t.
5. For each new segment B_i of the spline:
   a) If $k > 0$, let $B = B_i, k = k - 1, P$ be the control polygon of B_i. Go to step 4.
   b) Else add segment to A.
6. A is the final approximation. Terminate.

The heuristic of the algorithm guarantees the best subdivision option for each step. Hence, the algorithm does not have to execute any recursive search. However, the result curve can still exhibit inconsistencies in offset distance especially for a small number of iterations.

2.5.2. Polygonal approximation offset approach

Polygonal offset curve approximation method is based on manipulating a polygonal approximation of a curve instead of its control points. Comparing to the alternatives this approach has several advantages and disadvantages.

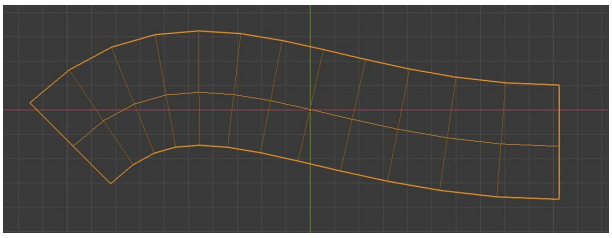


Fig. 16. Offset approximation by applying a simple bevel object to a Bézier curve in Blender 2.8

The main disadvantage of the method is the limited information about the offset curves. The new polygon is completely reliant on the original curve which may result in noticeable loss of visual quality. The method can also limit the ability to further modify offset curves. It should be noted that for many purposes a polygonal approximation of a curve is not enough.

A big advantage of this approach is the relative increase in computation efficiency. For the purposes of displaying an offset curve as well as many others it must be transformed into a polygon first. Calculating the polygon right-away allows to skip a step and

save computation time. Another advantage is the invariant precision. Without any additional restrictions for the offset points positions they can be placed exactly on the true offset curve.

Naive point-wise offset

The idea behind naive offset curve approximation is to simply apply offset transformation to the polygon representing the original spline. The offset distance is constant and is equal to a given value. The offset vector for each point is equal to normal vector of the original curve in the corresponding position.

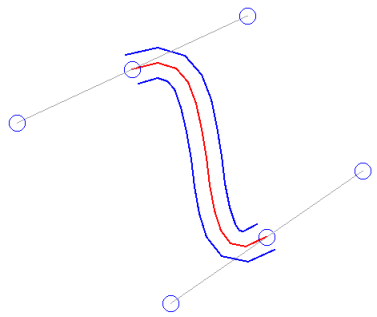


Fig. 17. Naive point-wise offset applied to a cubic Bézier segment

Let $B : [a, b] \to \mathbb{R}^2, B(t) = (x(t), y(t))$ be the original spline, $d \in \mathbb{R}$ be the offset radius, vector $t = \{t_1, ..., t_n\}$ be vector of evaluation positions, $P_0 = \{p_i | \exists t_i \in t : B(t_i) = p_i\}$ be a polygonal approximation of the spline, and $N = \{n_i | \exists t_i \in t : n_i = \frac{(y'(t_i), -x'(t_i))}{\sqrt{x'(t_i)^2 + y'(t_i)^2}}\}$ be the offset vector set. Then $P_1 \in \mathbb{R}^{n \times 2}$ defined the following way:

$$P_{1i} = d(P_{0i} + N_i), i = 1, ..., n$$

is the polygonal approximation of the offset curve. This operation is equivalent to evaluating the true offset curve in predefined positions t_i.

Advanced point-wise offset

A substantial issue of every reviewed offset method is overlapping geometry. It occurs when the original curve bends too sharply so that there is not enough distance to fit the offset curve. This can result in visible artifacts (figure 18). This section provides a solution for this issue based on the naive point-wise offset method.

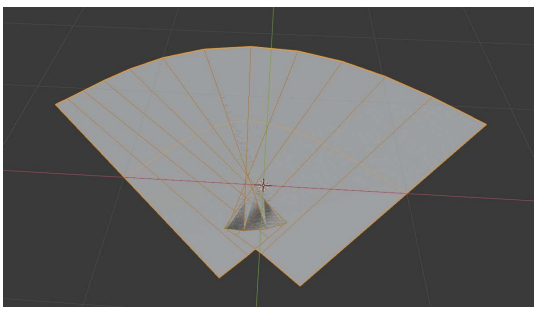


Fig. 18. Visible artifacts due to overlap (screenshot taken in Blender 2.8)

Let $B : [a, b] \to \mathbb{R}^2$ be a Bézier spline and $O : [a, b] \to \mathbb{R}^2$ be its offset curve. An overlap occurs when the following condition is met:

$$\exists t_1, t_2 \in [a, b] : (B(t_2) - B(t_1)) = q(O(t_2) - O(t_1)),\ q < 0.$$

In order to fix overlaps the following algorithm can be applied:

1. Let $B : [a, b] \to \mathbb{R}^2$ be a Bézier spline.
2. Let $P_0 \in \mathbb{R}^{n\times 2}$ be the polygonal approximation of B.
3. Let $P_1 \in \mathbb{R}^{n\times 2}$ be the result of naive point-wise offset applied on P_0.
4. Let $n \in \mathbf{N}$ be maximum search distance.
5. For each edge $(P_{0i}, P_{0(i+1)})$ of polygon P_0:
   a) If vectors $P_{0(i+1)} - P_{0i}$ and $P_{1(i+1)} - P_{1i}$ do not align:
      i. check n preceding and n consequent edges of P_1 (i.e. $(P_{1(i+j)}, P_{1(i+j+1)})$, $j = -n, ..., n$) to find a pair of edges intersecting in point I.
      ii. Set all points $P_{1(i-j)}, ..., P_{1(i+j)}, j = 1, ..., n$ to coincide with I
   b) Else continue to the next edge.
6. P_1 does not contain any overlaps. Terminate.

This algorithm replaces problematic geometry in the initial offset polygon with a triangle fan (figure 19).

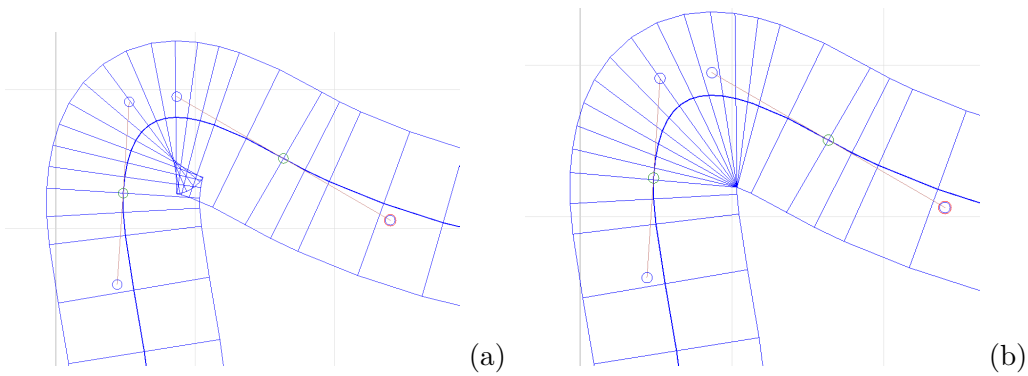


Fig. 19. Overlap reduction algorithm applied on a cubec Bézier spline (a - before fix, b - after)

The main downfall of the algorithm is its performance. If the offset curve does not contain any overlaps the algorithm finishes in linear time. But if there are overlaps, the time complexity becomes quadratic against the number of edges in polygon P_0.

The method is best suited for curves that will consequentially be exported into a polygonal mesh. It allows to save the offset curve into a geometry mesh supported by ASE format without any further evaluations. The result polygon does not aim to approximate the offset curve. This approach will be used in the project over any other method of curve offset.

2.6. Conversion from a parametric curve to polygonal mesh

Cruden Panthera does not support import of parametric curves into a track scene and is strictly limited to mesh geometry objects only. For this reason the track geometry must be converted into mesh in order to be used in Panthera.

Different software manages the geometry data organization differently. A majority of 3D editors allow an arbitrary number of adjacent edges for each point. While it provides the necessary flexibility, it can often lead to data redundancy for objects with a more uniform geometry structure.

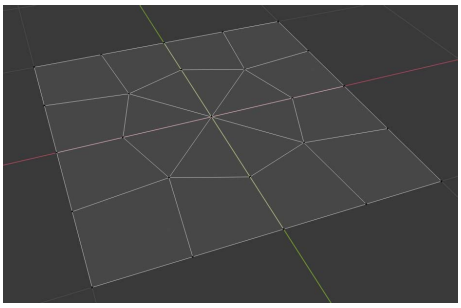


Fig. 20. Example of vertices with arbitrary amount of adjacent edges (Blender 2.8)

The process of conversion a Bézier spline into mesh varies from software to software, but it generally includes two steps: to generate a polygonal approximation of the spline, and to convert the polygon into the internal mesh format. This operation is trivial with a single spline, but it is not straight-forward with multiple offset curves that must be converted into one continuous mesh. The number of possible ways to connect two polygons into one mesh is exponential against the number of vertices in both polygons.

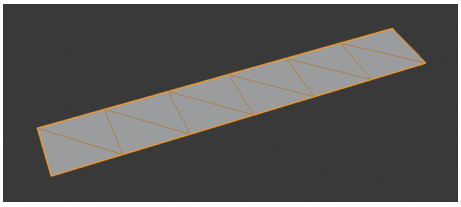


Fig. 21. Example of a face loop (Blender 2.8)

An advantage of using point-wise offset approach for generating offset curves is that the result curve polygon has the same number of points as the original. On top of that each offset point clearly corresponds to a point on the original curve. This allows the track geometry to be fully described as a single face loop (i.e. a sequence of quads where each quad shares a pair of neighboring vertices with the previous quad and two different vertices with the next one - figure 21). This makes the conversion to mesh a trivial task.

2.6.1. Texture coordinates and material data

Besides the spacial position of vertices in the track mesh a substantial part of geometry information concerns the appearance of the track. Different parts of the mesh can be rendered differently based on the assigned materials. The mesh color can be sampled from image files called textures based on the predefined correspondence between parts of a texture and mesh surface.

Texture coordinates of a vertex are a vector $t \in \mathbb{R}^2$. A single vertex may have one or more corresponding points in texture space. For this reason texture positions are grouped in triangles and the faces in the mesh are then coupled with them. The process of generating texture coordinates for existing mesh is called UV-unwrapping.

There are no explicit limitations for general texture coordinates to fall within the square $[0, 1]^2$. Even though the texture has a limited window of coordinates, the sampling during rendering is most commonly performed using texture tiling (i.e. a function $f : \mathbb{R}^2 \to [0, 1]^2$).

The most straight-forward approach to unwrap a mesh object is to map every quad in the mesh to a default quad. The default quad is usually selected to be a square with a corner at point $(0, 0)$ and a corner at point $(1, 1)$, however a different shape can be selected. This unwrap method is very limiting and is most often used for testing purposes.

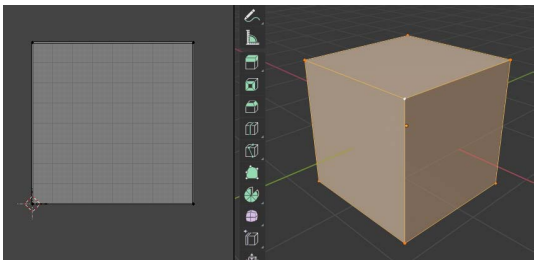


Fig. 22. A cube with a default UV-map (Blender 2.8)

Another approach used to unwrap road meshes is to take advantage of texture tiling. It uses a one-directional seamless texture of a road including markings. The geometry of the road is then unwrapped so, that the faces only exceed the boundaries of the texture along its seamlessness direction (figure 23). This method provides the ability to modify the shape of the road while preserving the relative position of the markings. However, a sharp turn in the road shape can cause noticeable texture stretch.

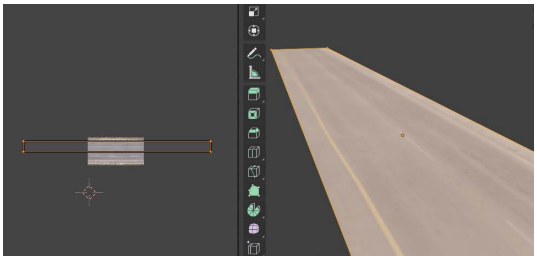


Fig. 23. Example of a road with one-directional seamless texture (Blender 2.8)

An alternative way to generate texture information for a road mesh is to combine multiple materials. This requires more complex geometry because the material data is stored per face. Each material is mapped to a seamless texture representing a type of road surface (e.g. asphalt, white markings paint, yellow paint, etc.). The road markings are then modeled according to the requirements and assigned a corresponding material.

The mesh can be unwrapped in an arbitrary way utilizing the texture seamlessness. A good way to unwrap a mostly flat road surface that also allows to automatically set the correct texture scale is to use x and y coordinates of the mesh points (given that z represents the vertical direction) as the texture coordinates. The markings can then be modeled without any reliance on the texture.

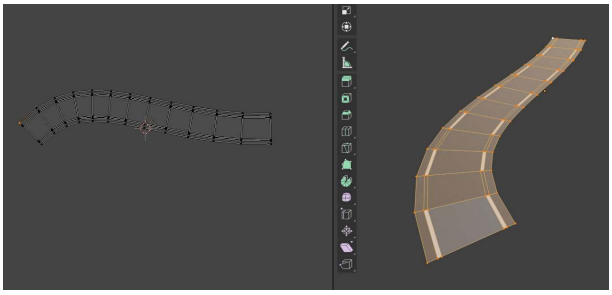


Fig. 24. Example of a road with manually modeled markings (Blender 2.8)

This approach has several advantages and disadvantages. As it was stated earlier, the approach requires more faces in the mesh. The level of detail of the markings is limited by the detail of the geometry and adding relatively small makings detail can lead to a significant increase in the number of faces. It also limits the selection of textures that can be used for the road. The textures must be seamless and directionless. Any directional pattern must be modeled with faces. As a consequence there is no efficient way to visualize road deterioration. It also means that the borders of all markings are sharp with no mix gradient between the neighboring materials.

An advantage of this method is the complete lack of texture stretch. All the markings are stored as polygons and do not have to be pixelated before rendering. It also allows users to create any markings pattern without the limitations of a finite set of photo textures. Another benefit is the straight-forward approach to the texture scale. The scale of the texture pixels stays invariant to the scaling of the mesh. The main advantage of the approach over the alternatives is its automation simplicity. Generating texture coordinates is a trivial task, and material information can be easily added to the profile function of the track. This approach to generating road texture data will be used in the project.

2.7. Analysis conclusion

To conclude there are multiple important takeaways from the main requirements analysis. A race track can be described as a parametric function of the position along its middle line. The function can be decomposed into two 2D functions: the track layout and profile. The layout can be defined as a Bézier spline, which satisfies the formal requirements and has high usability. The profile component function can be described with a polygon extruded along the layout curve. The profile must also define information about materials. Advanced point-wise curve offset method can be used to ensure clean geometry after the profile extrusion. The track geometry is mapped into texture space using a trivial transformation and utilizes multiple materials to create custom road markings.

The track must be exported into ASCII Scene Export format as a polygonal mesh. Cruden Panthera does not support importing any types of parametric curves. The additional configuration files can be included for users to copy into the desired folder manually and do not have to be generated for every track.

3. Race Track Editor application design

After analyzing the main functional requirements, the application can be designed. The application design is a continuous process. The final application's image can change during the development and implementation. For this reason it is important to incorporate regular application design evaluations in the implementation workflow. The main aspects of Race Track Editor design are the graphical user interface and the interactions with Cruden Panthera.

The usual application development pipeline relies heavily on the market research and user tests even at its earliest stages. However, this project does not aim to create an application targeted at a general audience. Race Track Editor is only supposed to fit the workflow of the Smart Driving Solutions research center, which only occupies less then ten people.

Cruden Panthera is another reason for how limited the focus group of the project is. The software does not have a big user base and only a small portion of Panthera users may be interested in creating custom race track scenes. For these reasons the target group research in this project is relatively minimal.

It should also be noted that the application's design defined in this chapter does not represent a finished application. It is closer to the initial iteration of user interface development process and the final Race Track Editor implementation requires further testing in order to improve its interface, even though the majority of its functional requirements are met.

In this chapter the stages of the application design are described in further detail.

3.1. Design requirements

Besides the functional requirements, there is another set of guidelines that influence an application development process - design requirements. While functional requirements are mostly focused around the program's output, these design guidelines have to do with the application workflow. They are effected by a wide range of parameters such as the target group physical capabilities or the environment the application will be used in. Those factors do not have to change the program's outputs, but they can dramatically impact the way the application is interacted with.

As it was stated earlier, the target group of the project is very small and does not allow for any meaningful statistical analysis. However, it is still important to account for the specifics of their workflow. This is achieved by personal communication with the group. It also helps outline the main goals of the application and the priorities among features. The most basic guidelines can be stated without any investigation: the researchers primarily use desktop computers with the usual set of input controls (i.e. a keyboard and a mouse). The main operational system is Microsoft Windows 10. Other design requirements are rarely provided explicitly and often have to be defined during the data collection stages of the development.

3.1.1. Usage goals

Studying usage goals is a good way to form an understanding of design requirements based on the functional requirements. It may also provide finer details for the functions and features of the application. A usage goal is a goal a user may want to achieve while using the application. The usage goals for Race Track Editor are:

- Create an arbitrary race track.
- Create a recreation of an existing track.
- Create a set of obstacles according to a standard description.
- Modify a previously created track for different parameters.
- Generate border definitions for a previously created track.

These usage goals suggest several additional features for Race Track Editor besides the main functional requirements. A significant addition to the initial concept of the application is the ability to specify the position of the track layout curve and track parameters precisely and in metric units. This ability can make recreating existing tracks and creating obstacle courses by the standard descriptions (figure 25 - the standards are often defined against the parameters of the vehicle) much easier.

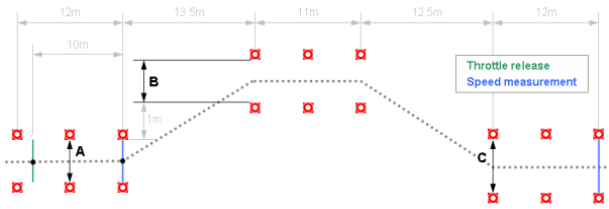


Fig. 25. ISO double lane change test specification [20]

Some of the features suggested by the usage goals such as the ability to add different obstacle objects besides the main track geometry and the ability to display reference images on the background of the editor window were not added into the design due to the lack of time.

The border definitions mentioned in the goals is a way to connect the track to the physics model used by the research group. The model does not utilize the built-in Panthera collision detection system and has to determine whether the car has left the track boundaries or not on its own. For this to happen it also has to have information about the track. The ability to generate this data is a crucial functional requirement for the workflow of the research.

3.1.2. Additional requirements

The application is developed as a light-weight replacement for big 3D editors such as 3Ds Max or Maya in the race track scene creation process. Because of its narrow purpose Race Track Editor does not have to have a huge variety of tools and functions for advanced polygonal modeling and complex texturing. The minimalist set of functions and tools plays to the application's advantage in more than one way.

One of the reasons there is a need for the application is how crowded and difficult to learn the mainstream 3D editors' user interfaces are. The vast majority of 3D editing software has made the decision to sacrifice intuitive user interface for the diversity in

features. Despite continuous improvements in their interface programs like Blender or 3Ds Max remain in the constantly contracting group of applications that require reading its user manual before use. There are several reasons for why this is the case, but the most important is the huge spectrum of features the programs provide. Race Track Editor can include every necessary function and still be less crowded.

Another reason the mainstream 3D editors tend to have a complicated user interface is the three-dimensional view. Many operations that are intuitive and do not need any additional UI elements to telegraph or influence their consequences on a flat plane (e.g. dragging files with a mouse movement in the file explorer) require additional ways of interaction in three dimensions. Even though the general approach to navigate in 3D view does not tend to vary significantly, many 3D editors use their own UI systems for it. This goes to show yet another problem with big 3D software - the lack of universally recognized conventions, which in turn extends the learning period. In Race Track Editor these problems can be avoided by reducing the user input into two dimensions.

Because of that the user interface of Race Track Editor must be designed with a minimalist approach to UI elements in mind. All the direct manipulation operations must be limited to two degrees of freedom at a time (i.e. reduced to two dimensions).

3.2. Graphical user interface design

The design of an application's graphical user interface is an iterative process. It aims to create an intuitive UI layout while avoiding having to implement the application at every iteration[21]. The iterations of the design process include several steps: collecting information about functional and design requirements, creating a prototype of the user interface, and the prototype evaluation. The first step is repeated in every iteration, because the prototype testing may reveal additional information that was not initially known. The evaluation step may be executed in a number of ways. The most common approach is to perform user tests, but it is sometimes replaced with a cognitive walk-thought. The process terminates whenever the layout satisfies every requirement and no new requirements can be found during prototype evaluation and analysis.

As it was stated earlier, Race Track Editor user interface design is at an early iteration of the development process. The UI layout was initially designed as a paper mock-up (i.e. a set of pictures and text roughly describing the position of the elements and the ways of user interaction - figure 26). This stage of the design is usually tested with users, but a simple cognitive evaluation was applied mostly due to the lack of time.

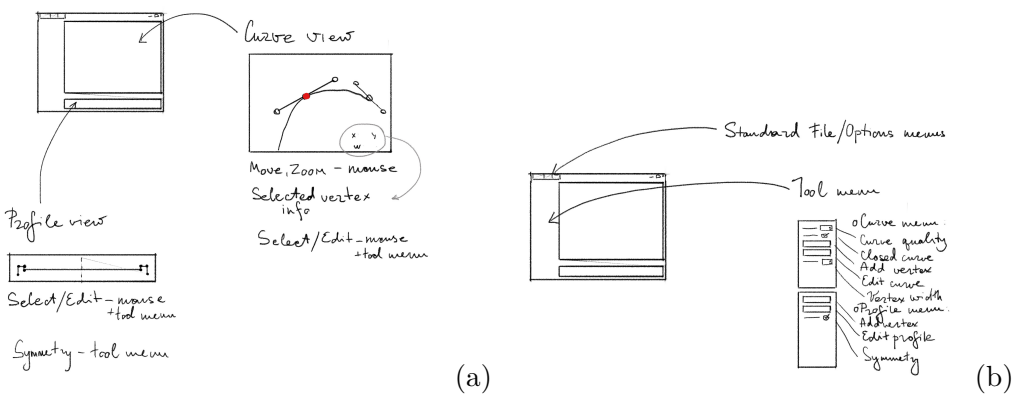


Fig. 26. The paper mock-up of the interface

The evaluation of the mock-up revealed several issues with the layout. The tool menu had to be moved to the right side of the screen to mimic the conventions used in some graphical editors such as Adobe Photoshop 18 and Blender 2.7 and later. It also showed the need to provide some spacial differentiation between track parameters and spline modification tools.

Following the paper mock-up an interactive prototype is often developed to ensure a more precise requirements definition at an early development stage. This can be a time-consuming step for programs that use direct manipulation interactions[22] because those systems can be difficult to set up in a simple prototype. Due to the lack of time an early implementation stage was utilized for the evaluation (figure 27).

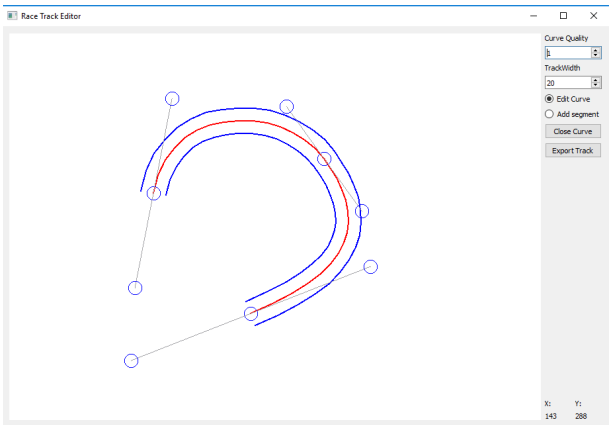


Fig. 27. The early implementation stage of Race Track Editor tht was used for the evaluation

The main focus of this stage's evaluation step was on the Bézier splines, which proved to be a powerful and intuitive modeling tool. However, during the evaluation the need for additional tools to modify the spline was discovered. Another evaluation discovery is the need for action history. Direct manipulation interfaces often rely on a system to undo and redo user actions; and Race Track Editor would benefit from introducing such system. The prototype did not provide any way to test a big portion of the program features, which must be tested and evaluated from the usability standpoint in the future.

The final UI layout (figure 28) uses the discoveries of the previous design stages, but many features have to be tested further.

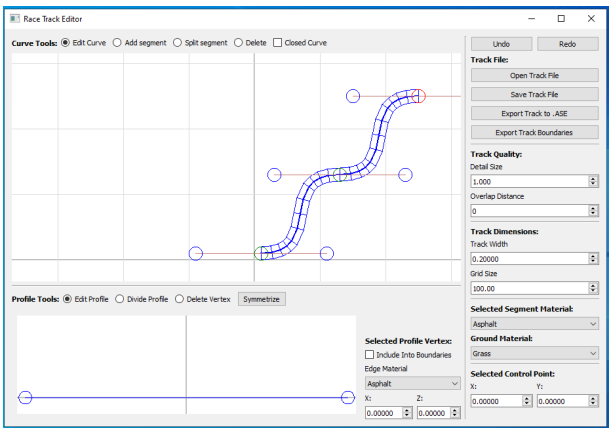


Fig. 28. The user interface of Race Track Editor final implementation

3.3. Cruden Panthera import pipeline integration

Another reason there is a need for the application is Cruden Panthera's import pipeline. Panthera is an independent software originally developed by a small team. Because of that it suffers from the shortcomings many other independent programs have as well, such as the lack of user centered design approach. Many features of Panthera are technically functional, but require a lot of manual work by user.

The process of importing a track scene involves using a separate program, creating or copying configuration files by hand, converting the input scene file into geometry source files, and importing the geometry sources into TreckEd. After that configurations adjustments may be needed, which in many cases means rewriting contents of configuration files by hand. In case of an error the user is not notified directly. The error message can be found in a log file in a different folder.

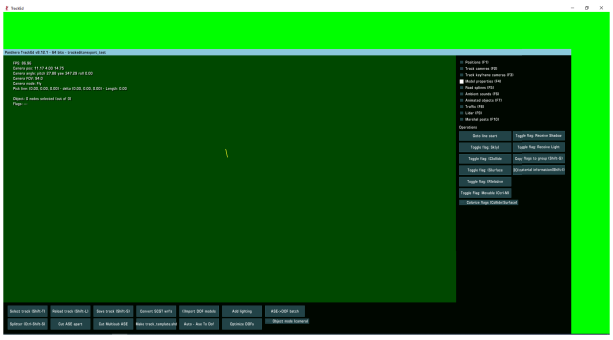


Fig. 29. TrackEd does not support full screen mode

A big portion of Panthera's usability issues could potentially be solved by adding a more user-friendly way to interact with the software. Unfortunately, there is no way to change the software from the inside. The usability can only be improved from the outside, i.e. in order to make TrackEd more usable Race Track Editor must automate as many import steps as possible. However, the automation possibilities are limited due to the technological reasons. There is no way to export a track from Race Track Editor into the internal binary geometry source format, therefore TrackEd is the only option for the task.

Race Track Editor can utilize Panthera's segregated configuration files system. It can provide the required files and folders minimizing the user's interaction with TrackEd to the simple input scene conversion into geometry source files.

3.4. Design conclusion

To reiterate the design stage takeaways, the graphical user interface design of Race Track Editor is at an early stage of development. The UI layout could be improved with further user tests. The list of necessary features has been extended during analysis iterations, but some of them must be implemented in the future.

Race Track Editor integration with Cruden Pathera provides a limited improvement to the import pipeline usability. Along with the track scene the application can provide a package of configuration files, reducing the necessity for the user to interact with TrackEd.

4. Implementation of Race Track Editor application

The implementation stage of the project is tightly intertwined with the design process. As it was shown in the previous chapter, the early versions of the application were used in the design iterations as prototypes to test user interface on. This is a common practice in application development process, though the exact design iteration used for the first implementation varies based on multiple parameters.

The more developed design of the first implementation generally leads to better code organization and easier maintenance. This comes at the price of the extended development time as less processes can be worked on concurrently. It also requires developing more prototypes that will not be utilized in the long run. While this is not a significant issue for a wide range of form-filling applications, for many direct manipulation programs it is.

The architecture of Race Track Editor was selected to follow a simple MVC design pattern. This pattern divides the program functionality into three interconnected parts: the Model is responsible for the inner logic (often referred to as business logic), the View concerns the user interface logic, and the Controller ensures the communication between the parts [23]. The controller is also responsible for handling user input. Unlike some of the alternatives, the pattern is best suited for relatively small applications and does not guarantee a strict segregation between the individual parts. This can lead to some maintenance issues, but it also provides the flexibility to create arbitrary direct manipulations interfaces.

Due to the limited development time the first implementation was introduced early on. This led to some overlapping in the roles of the model and the controller as more features were added on top of existing code, however, this issue can be resolved with simple code restructuring.

In this chapter the final Race Track Editor implementation is revised.

4.1. Technologies used

There exists a big variety of environments and frameworks that allow to implement all the required functions of Race Track Editor. They differ in flexibility and robustness as well as native platforms, programming languages, and integration of other technologies. Every option has its advantages and disadvantages.

Besides the most popular frameworks used in the industry (e.g. .NET framework) a viable option for Race Track Editor implementation is Blender internal user-plugin system. It allows to create custom user interface elements inside Blender and utilize the wide spectrum of Blender internal modeling tools. The plugin scripts are written in Python and use Blender API to access its features. Blender has a rich history of integrating user-made elements into the software and almost every aspect of the program can be utilized and customized via scripts (e.g Retopoflow - figure 30). Blender does not limit the possibilities of Python, so many plugins take advantage of different programming languages integration into Python.

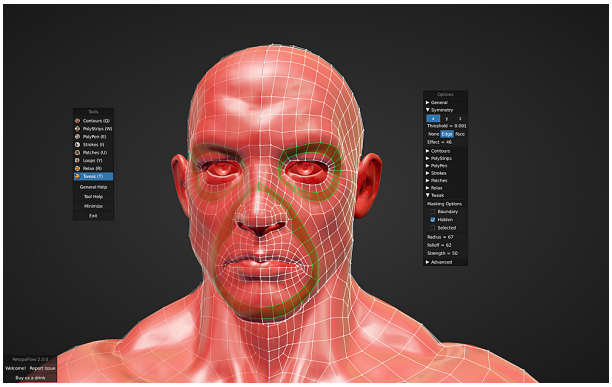


Fig. 30. Retopoflow - an advanced Blender plugin with custom UI [24]

Implementing Race Track Editor as a Blender plugin can have its benefits and downfalls. The main advantage of implementing Race Track Editor inside Blender is the wide range of features the project can utilize. The editor view navigation system, the user action history system, Bézier splines support as well as ways to save the track files are among the primary functional requirements of Race Track Editor.

The main disadvantage is the contradiction with the project design requirements. One of the application's goals is to replace the big 3D editors in the track creation process. While the plugin can streamline the track editing and export, it does not eliminate user interactions with Blender completely and would still require a certain level of familiarity with the software in order to be used. Another disadvantage of this implementation approach is the necessary maintenance. Blender is a rapidly changing program, which does not usually sustain backward compatibility. This can potentially result in the need to update the plugin for every new Blender version.

There also exist ways to implement Race Track Editor inside other 3D editors that allow user plugin integration. While the details such as programming language and the set of accessible features may vary, the main advantages and disadvantages stay similar to Blender. For this reason the project utilizes a more standard approach to use a general programming framework for Race Track Editor implementation.

Qt was selected as the implementation framework based on a number of parameters. It is a free and open application development framework that provides functionality to create applications with custom user interfaces [25]. Qt uses C++ programming language, which allows for more computation efficiency and different technology integration comparing to some alternatives. Another advantage of Qt is that its applications do not rely on OS-specific libraries and can be compiled for different platforms with minimal changes in source code.

4.2. The application structure

The project's source code structure is dictated in part by the selected development framework. Qt requires custom widgets (in Qt terminology UI elements are called widgets) to have code-behind behaviour and appearance definitions. Different parts of the project are connected with Qt event system, which allows to trigger predefined functions in response to input events as well as update widgets from code.

Every Qt application has a main window class. It is instantiated in the program's entry point and is rendered on the screen. The main window class contains all its child widgets including custom widgets. The widgets defined inside Qt Creator Design mode

(the definitions are stored in XML file called "mainwindow.ui") are connected to the application during compile time. For this additional class definition source files are generated and included in the compilation bundle.

Race Track Editor application uses two custom widgets: layout curve editor widget (defined in `RenderArea` class) and track profile editor widget (`TrackProfileArea` class). Those are instantiated in the main window constructor and are then added into the set up UI layout. The constructor also takes care of the event communication system setup. Every object can send and receive signals using the predefined signal and slot attributes respectively. In order for an object to react to an event its slot has to be connected to a signal of the event sender. There are no limitations to how many connections are allowed per signal and per slot.

As it was stated earlier, the project does not follow the selected MVC design pattern strictly. From its standpoint the controller role is divided between the main window class, the curve editor class, and the profile editor class. The custom view logic is also handled inside custom widget classes as well as a portion of business logic (figure 31). This code organization is not optimal and may lead to maintenance issues, however, this problem can be fixed with relatively simple code restructuring.

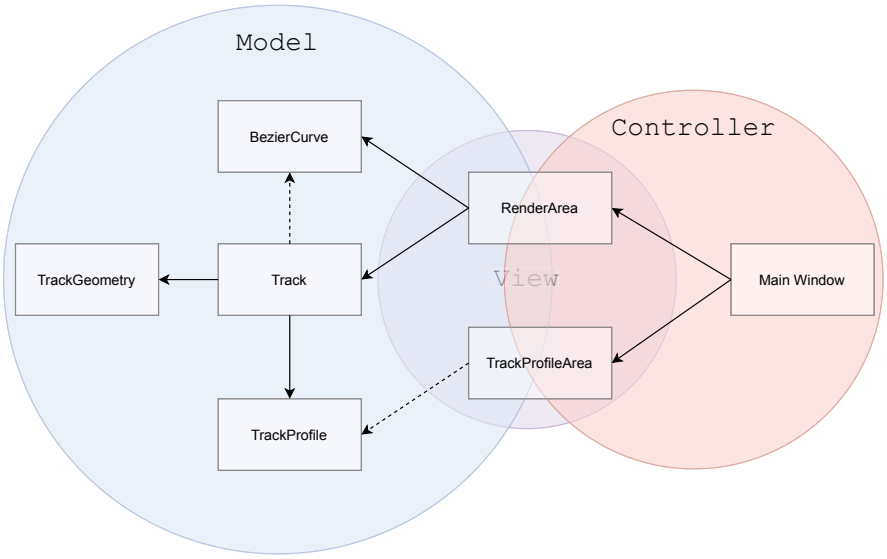


Fig. 31. The application class hierarchy against MVC pattern (solid arrows point from parent to child classes, the dashed arrows denote communication via pointers)

The algorithms and methods described in the analysis chapter are implemented in the business logic application section. The general Bézier spline is represented in the class `BezierCurve` (beziercurve.h). It keeps track of all its segments and the cumulative control polygon. The general Bézier segments are defined in `CurveSegment` (curve.h). Cubic Bézier segments are described in `CubicBezier` (cubicbezier.h), which inherits from the general curve segment class. It contains the implementation of the recursive De Casteljau's algorithm (in function `deCasteljau`) used to convert the spline into a polygon. The implemented version is extended to include naive point-wise curve offset algorithm - the calculated polygonal approximation of the spline is automatically offset at a given offset distance.

The continuity of the track layout spline is ensured by class `BezierControlPoint` (beziercontrolpoint.h). It represents triplets of neighboring points in the spline control

polygon, and modifies the entire triplet based on the transformation in one of the points. The class ensures C^1 continuity according to the method described in the dedicated section of the analysis.

The track profile definition is located in file trackmesh.h in class `TrackProfile`. It contains the profile polygon points and the material data in form of identification numbers. `Track` class combines the Bézier spline and track profile. It contains methods to generate geometry mesh from a spline and a profile polygon. The advanced point-wise curve offset algorithm is implemented in the function `resolveGeometryOverlaps`. The function takes 2 polygons and a threshold as parameters. The first polygon serves as the main curve, the second polygon is then effected by the overlap reduction.

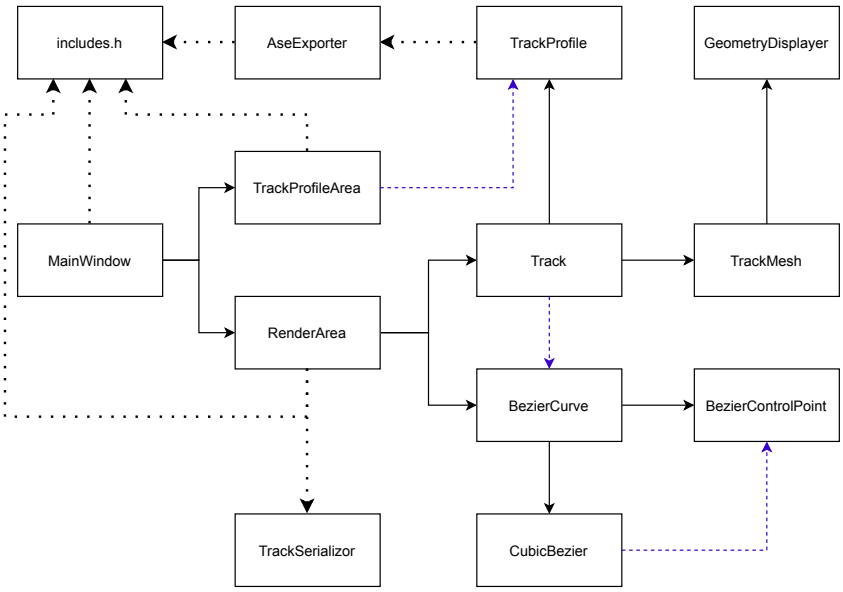


Fig. 32. The application object structure (here solid arrows denote relation "A instantiates B", purple dashed arrows represent "A refers to B" relation, and dotted arrows stand for "A uses static members of B")

`TrackMesh` class (trackmesh.h) contains the mesh representations of all the objects to be displayed as instances of `GeometryDisplayer` class (trackmesh.h). It is used to display the layout curve and the actual track geometry simultaneously. In `GeometryDisplayer` the set of all the mesh vertices and edges with corresponding material IDs is stored. An instance of `GeometryDisplayer` can be converted into ASE format string by the static method `ASEFormat` defined in the `AseExporter` class (Exporter/aseexporter.h). It uses the track geometry parameters to define a universal face loop index structure, which is then applied to the mesh. The method also includes predefined geometry objects such as the skydom and the ground plane into the exported scene string.

The majority of the user interaction logic is defined in the `RenderArea` class (renderarea.h) and `TrackProfileArea` class (trackprofilearea.h). `RenderArea` handles all the layout editor widget mouse event responses, which include view navigation, moving the control polygon points, adding a new segment to the end of the spline, adding new knot points in-between existing knot positions, and deleting control points. The class instantiates `Track` class and performs all the operations on it. The logic of

the action history system is also defined here as well as "Save File" and "Open File" dialog calls and export dialog calls.

`TrackProfileArea` class takes care of editing the track profile polygon. It references the `TrackProfile` instance of the relevant `Track` object. The functions to add a point to the polygon, delete a point, move the selected point, and symmetrize the polygon by copying and flipping its left part against the middle line are found here as well. The function to change the edge material IDs is also located here. The material change is applied on the right adjacent edge to the selected point. The profile editor widget reacts to events send by the `RenderArea` object, which ensures applying implicit changes to the profile by such operations as "Undo", "Redo", and "Open File". In turn it sends events to inform the curve editor widget of profile changes to render the relevant geometry and record user actions in the action history.

Both `TrackProfileArea` and `RenderArea` also contain reaction definition to paint events. These arise whenever the application window is updated, i.e. as a reaction to the OS window event or an internal `update` function call. The classes utilize Qt standard draw commands to render the polygonal data as a set of geometrical primitives (i.e. lines representing the polygon edges and circles for interactive elements).

The application has a set of predefined mesh materials `global_materials`, which can be found in includes.h file. The material IDs used throughout the project reference the materials in the list. The application provides a way to save the track data in a text format as a set of numbers corresponding to the individual track properties. The static methods to load and save the track in the format are located in `TrackSerializor` class (Exporter/trackserializor.h).

4.3. Output formats

The application provides several output file format options. The first is the required ASE file. It follows the syntax rules described in the analysis chapter. The material section is represented with a single material node and multiple sub-materials, which only differ in names and ID numbers. The set of sub-material definitions is hard-coded in file includes.h. The following section of the output file is a "*GEOMOBJECT" corresponding to the track geometry. The next section is the definition of a simple plane that represents the ground. The last geometry definition is of the skydom object. It is a polygonal sphere mapped to an environment texture by default. The hard-coded geometry object definition can be found in file skydomGeomObj.h.

Another output file format describes the parameters of the track scene to be reused inside Race Track Editor. Each line of the file contains an ordered set of integers or real numbers. There are no syntax restrictions or tag system. The format is not sensitive to white spaces. The role of each number in the file is defined by its position in the set of all the numbers.

Listing 4.1 The default track scene defined in the internal Race Track Editor Format

```
2
0
-90.000000 10.000000
10.000000 10.000000
110.000000 10.000000
30.000000 130.000000
130.000000 130.000000
230.000000 130.000000
150.000000 250.000000
```

```
250.000000 250.000000
350.000000 250.000000
20.000000
0
2
0.000000 0.000000
10.000000 0.000000
1
0
100.000000
1.000000
0
0
0
0
4
```

The first integer in the file represents the number of segments in the track layout spline. The second corresponds to a boolean value whether the spline is closed or not. The following are the positions of individual points in the control polygon of the spline. For each instance of class `BezierControlPoint` there are three lines of real numbers tuples. Each number in the tuple corresponds to the x or y coordinate of the control point's main position or either handle positions. The order of the position definitions is the following: left handle, main position, right handle. The next real number is the track width. The next integer corresponds to the overlap reduction threshold. After that the profile definition is listed starting with the number of points in the polygon and followed by the coordinates of each point similar to the positions of the control points. The last set of integers are the material indices for every edge in the profile polygon and then for every curve segment. The last number in the file represents the ID of the ground material.

The last output file format option is CSV (comma separated values). It describes the track boundaries as a set of quads. Each quad is assigned a material ID based on the segment material ID. The first integer in the file corresponds to the number of quads. After that each line defines a single quad with 8 real numbers, each couple of numbers corresponds to x and y coordinates of a corner of the quad. The last integer in the line is the material ID.

Listing 4.2 CSV file describing an extremely simple track with only 2 quads

```
2;
0.100000; 0.000000; 0.100000; 0.200000; 1.570000; -0.010000;
    1.570000; 0.190000; 0;
1.570000; -0.010000; 1.570000; 0.190000; 3.150000; -0.010000;
    3.150000; 0.190000; 0;
```

The boundary quads are constructed based on the "Include Into Boundaries" profile vertex point properties. The edges of the quads perpendicular to the track layout curve connect neighboring boundary points.

5. Results

In this chapter the final implementation of Race Track Editor is used to generate several track scenes to showcase the application's capabilities. Every track generated in this chapter underwent the same set of operations to be imported into Cruden Panthera. After designing the track layout and profile it is exported into an ASE format file. The file is then imported into a track scene using TrackEd. The scene is saved which generates the lacking configuration files. The last step of the import is to modify "geometry.ini" configuration file to set the flags of the track object and the ground plane. This is needed in order for Panthera to recognize the objects as surfaces with which the car can interact. The corresponding property flag is 4 (listing 5.1).

Listing 5.1 The content of geometry.ini file for the first example race track

```
objects
{
  asphalt_0_track
  {
    file=asphalt_0_track.dof
    flags=4
  }
  grass_4_ground_plane
  {
    file=grass_4_ground_plane.dof
    flags=4
  }
  sky_mat_5_skydom
  {
    file=sky_mat_5_skydom.dof
    flags=0
  }
}
```

The first example of Race Track Editor output is a track created without any real-life references. It demonstrates the general capabilities of the editor (figure 33, 34).

The second example is a recreation of a real-life track - Aintree Circuit[26]. It also incorporates a more complex track profile definition to create road border markings (figure 35, 36).

The third track is a free-form open track. It serves as a demonstration of more complex road markings (figure 37, 38). This approach can be useful for obstacle course standards recreation. While it does not replace special obstacle geometry objects completely, it still can provide visual information about the course.

The last two track scenes are based on a real-life race track - Aspen Racing Circuit[27]. The forth example (figures 39, 40) showcases application of different materials per layout curve segment. The fifth track scene (figures 41, 42) combines material application per segment and per profile polygon edge. The fifth example track can represent the circuit where some parts of the road are wet and therefore have different friction parameters.

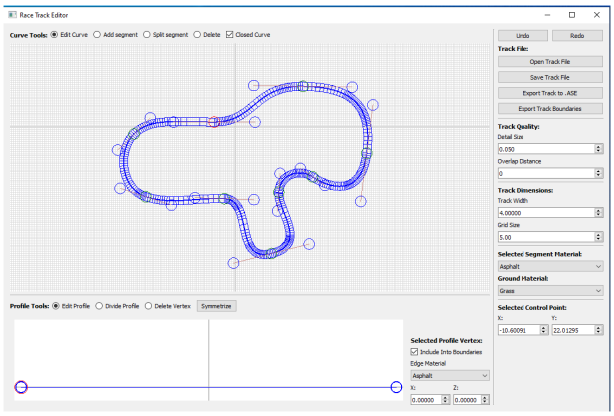


Fig. 33. A simple track created from scratch in Race Track Editor (appendix A)


(a)


(b)

Fig. 34. The first example track imported into Panthera (a - TrackEd view-port, b - simulation screenshot)

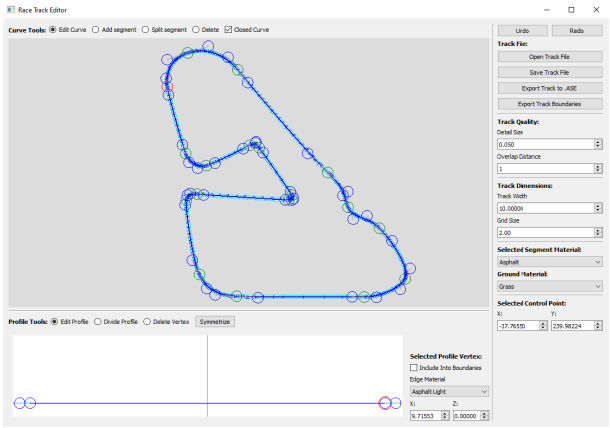
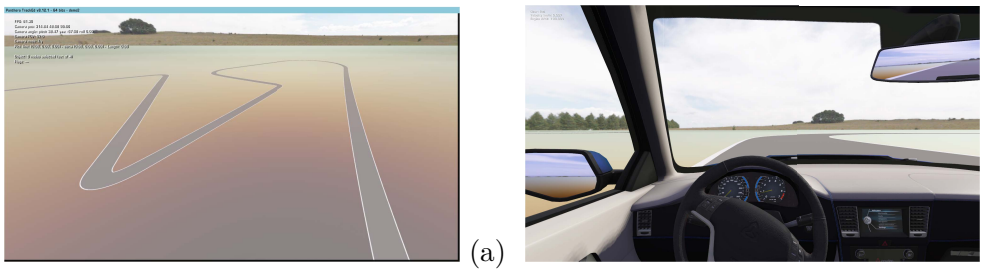


Fig. 35. Aintree Circuit in Race Track Editor (appendix A)

5.1. Known issues

There are several issues in the current implementation of Race Track Editor. Besides the user interface design, the behaviour of the user action history system is sometimes unreliable. This has to do with Qt event communication system; some events can cause signal feedback loops, which overwrite the entire action history. A more precise test case has to be developed to diagnose the issue.

Another problem of the implementation is the incorrect normal vectors in the output ASE file. The application uses the assumption that the track geometry is flat and applies a default vertical vector as a normal vector for every track mesh face. This



(a) (b)

Fig. 36. Aintree Circuit imported into Panthera (a - TrackEd, b - simulation screenshot)

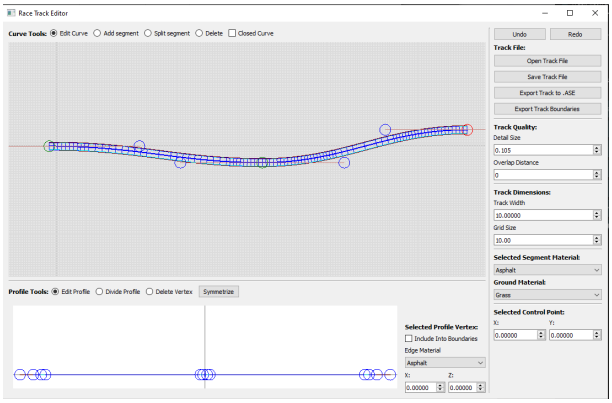
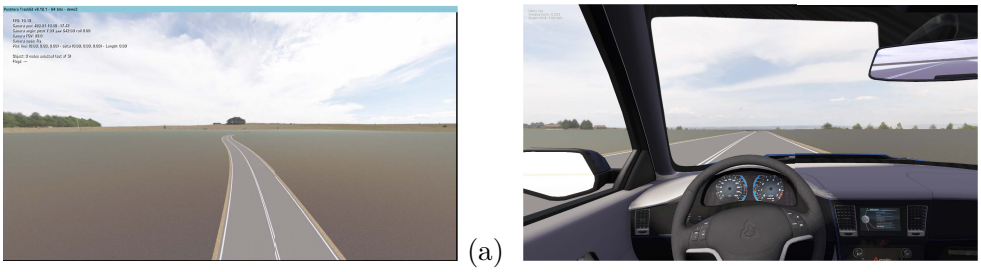


Fig. 37. The open track example (Race Track Editor - appendix A)



(a) (b)

Fig. 38. The open track example (a - TrackEd, b - simulation screenshot)

produces incorrect shading of the track geometry with a non-flat profile (figure 43). This can be solved by generating normal vectors for every edge of the profile polygon and then applying a transformation on the vectors for each face strip perpendicular to the layout curve. The approach has not been implemented due to the lack of time.

Another issue that needs addressing is the "Split segment" operation behaviour. The final implementation includes the operation to add control points to a curve in-between existing points. This operation does not correspond to a standard split operation described in the analysis chapter and does not preserve the spline shape, because the shape preservation would require transforming the curve from uniform to non-uniform.

These issues along with the graphical user interface of the application require further work in the future.

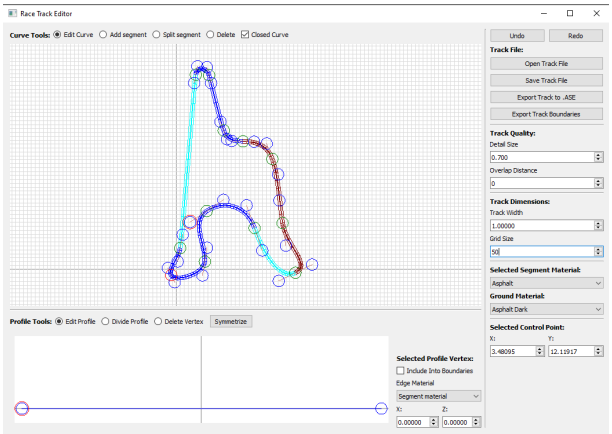


Fig. 39. Aspen Circuit - different segment materials (Race Track Editor appendix A)


(a)

(b)

Fig. 40. Aspen Circuit - different segment materials (a - TrackEd, b - simulation screenshot)

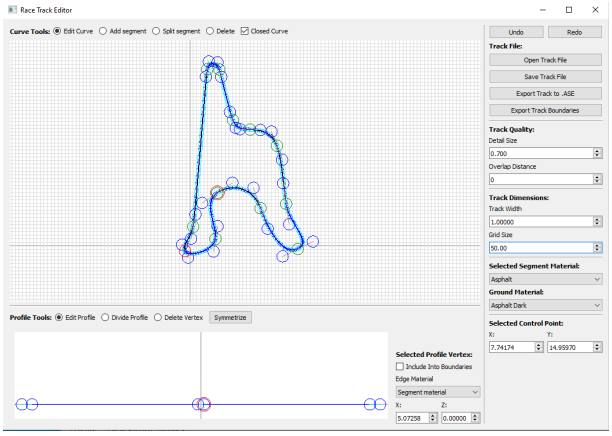
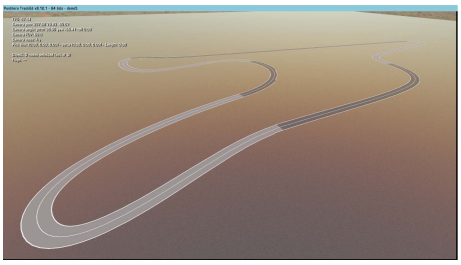
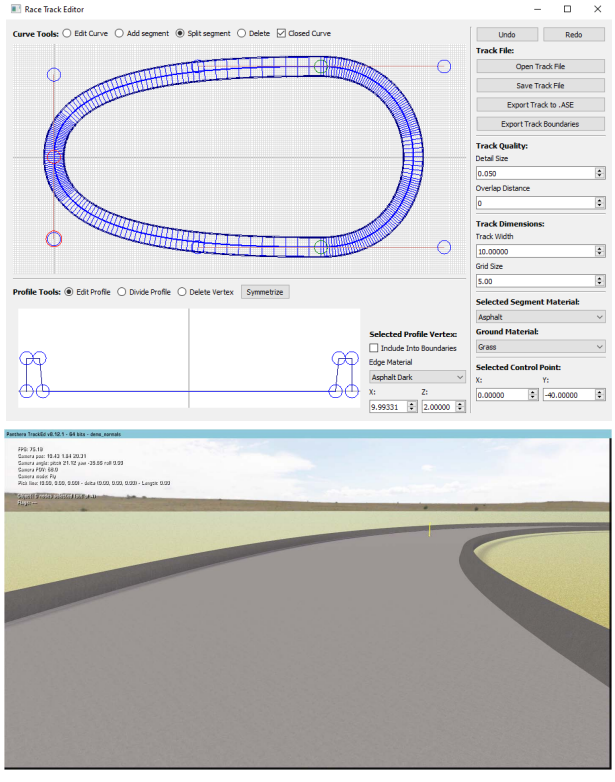


Fig. 41. Aspen Circuit - combination of profile and segment materials (Race Track Editor - appendix A)


(a)

(b)

Fig. 42. Aspen Circuit - combination of profile and segment materials ((a - TrackEd, b - simulation screenshot)



(a)

(b)

Fig. 43. Race Track Editor limited profile support: normal vectors are not generated properly (a - Race Track Editor, b - TrackEd)

6. Conclusion

The project was focused around the development of an application that would simplify race track scene creation process for the purposes of the Smart Driving Solutions research center.

In the first chapter a description of Cruden Panthera software package and its track scene import pipeline is provided. The specifics of ASCII Scene Export format and its usage in Panthera are studied and listed in the corresponding section. The following sections of the analysis chapter provide the definitions of the mathematical issues of race track geometry representation from multiple perspectives. The real-world standards were taken into account; and the problem of race track geometry definition was formalized as a parametric function of the position along its middle line. The function was then split into two component functions representing the track layout and the profile, both 2D functions.

The different mechanisms of layout definition were then assessed. The emphasis during the assessment was put on the usability of the method. Bézier parametric splines were selected as the track layout definition method because of the flexibility and versatility. The track profile component function was defined with a polygon representing the track cross-section. The information concerning the visuals of the track was also included in the profile representation.

The project studied the mathematics behind Bézier curves and different algorithms of curve offset, which was necessary to combine the track component functions. The advanced point-wise curve offset method was created based on the analysis of the existing approaches. The method is tailored specifically to work with Bézier parametric curves that will be subsequently converted into polygons and aims to reduce possible visible artifacts.

The next chapter described the design process the application went through before the final implementation. The iterative design approach was selected to lean the application towards the user-centered architecture. Several stages of the prototype design were presented in the chapter as well as the analysis of several early design iterations. The design of the final implementation is not optimal and further iterations are required to ensure better user experience.

The result of the analysis and design was Race Track Editor final implementation. Despite some counter-intuitive aspects of the user interface and issues with code structure, the implemented application satisfies the functional requirement of the project and adds important features to the initial idea of the application. The implementation details were described in the implementation chapter.

Finally, several output race track scenes fully generated with Race Track Editor were demonstrated. The demonstration also showed some of the weaknesses of the current implementation such as wrong normal vectors for 3D track profiles.

6.1. Future work

As it was mentioned earlier, the application needs further user tests and design iterations to enhance user experience and the efficiency of use. The current implementation also

includes some minor bugs which should be fixed. The full support of the 3D track profiles should also be added in the future.

Besides fixing the existing issues, the future work also includes the addition of the third dimension to the track layout curve. Another possible enhancement is a more sophisticated system of track width and profile definition that would allow create tracks with variable width and camber.

Appendix A.

CD contents

- `RaceTrackEditor_SourceCode.zip` - the project source code compilable in Qt 5.14.1;
- `RaceTrackEditor-release-x32.zip` - the launchable application for MS Windows 10 and user manual;
- Race Track Editor output demos (to run a simulation on a demo it has to be unpacked into Cruden Panthera tracks folder):
  - `RaceTrackEditor_demo1.zip` ,
  - `RaceTrackEditor_demo2.zip` ,
  - `RaceTrackEditor_demo3.zip` ,
  - `RaceTrackEditor_demo4.zip` ,
  - `RaceTrackEditor_demo5.zip` ;
- `TEX.zip` - LaTeX sources for this document with full resolution screenshots;
- PDF version of this document.

Bibliography

[1] *A simple ASE file of a square polygon*. 2005. URL: http://www.solosnake.com/main/ase.htm (visited on 05/10/2020).

[2] Paul Johnson. *What does 'a weird camber' mean when talking about a race track*. 2018. URL: https://www.quora.com/What-does-a-weird-camber-mean-when-talking-about-a-race-track (visited on 05/07/2020).

[3] *Chinese Grand Prix - F1 Race - Shanghai International Circuit | Formula 1®*. 2020. URL: https://www.formula1.com/en/information.china-shanghai-interational-circuit-shanghai.7ESK3ZQ5DHqihx3DpPZKWL.html (visited on 04/25/2020).

[4] Gilbert Strang (MIT) and Edwin "Jed" Herman (Harvey Mudd) with many contributing authors. *10.1: Curves Defined by Parametric Equations*. 2019. URL: https://math.libretexts.org/Bookshelves/Calculus/Map%3A_Calculus_-_Early_Transcendentals_(Stewart)/10%3A_Parametric_Equations_And_Polar_Coordinates/10.01%3A_Curves_Defined_by_Parametric_Equations (visited on 04/25/2020).

[5] Horacio Florez and Belsay Borges (June 6th 2018). *Scalar and Parametric Spline Curves and Surfaces, Topics in Splines and Applications*. 2018. URL: https://www.intechopen.com/books/topics-in-splines-and-applications/scalar-and-parametric-spline-curves-and-surfaces (visited on 04/25/2020).

[6] Steve Marschner. *CS4620/5620 Introduction to Computer Graphics*. 2013. URL: http://www.cs.cornell.edu/courses/cs4620/2013fa/lectures/16spline-curves.pdf (visited on 04/25/2020).

[7] Robert R. Snapp. *Computer Graphics coarse: Splines*. 2015. URL: http://cs.uvm.edu/~rsnapp/teaching/cs274/lectures/splines.pdf (visited on 04/26/2020).

[8] Jim Armstrong. *Catmull-rom Splines*. 2006. URL: http://algorithmist.net/docs/catmullrom.pdf (visited on 04/25/2020).

[9] Evgeny Demidov. *An Interactive Introduction to Splines*. 2017. URL: http://www.ibiblio.org/e-notes/Splines/Intro.htm (visited on 04/25/2020).

[10] Matthias Teschner. *Image Processing and Computer Graphics coarse: Curves and Surfaces*. 2020. URL: https://cg.informatik.uni-freiburg.de/course_notes/graphics_06_curves.pdf (visited on 04/25/2020).

[11] Marco Paluszny Hartmut Prautzsch Wolfgang Boehm. *Bézier and B-Spline Techniques*. Springer-Verlag Berlin Heidelberg, 2002.

[12] Mike "Pomax" Kamerman. *A Primer on Bézier Curves*. 2018. URL: https://pomax.github.io/bezierinfo/ (visited on 04/26/2020).

[13] C.-K. Shene. *CS3621 Introduction to Computing with Geometry Notes: De Casteljau's Algorithm*. 2011. URL: https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/Bezier/de-casteljau.html (visited on 04/26/2020).

[14] Neil Dodgson. *Bezier Curves*. 2000. URL: https://www.cl.cam.ac.uk/teaching/2000/AGraphHCI/SMEG/node3.html (visited on 05/26/2020).

[15] Chong Nyuk Sian. *Approximating Offset Curves by Rational Bézier Cubics and Quartics*. 2008. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.567.490&rep=rep1&type=pdf (visited on 04/29/2020).

[16] In-Kwon Lee Gershon Elber and Myung-Soo Kim. *Comparing Offset Curve Approximation Methods*. 2006. URL: http://www.cs.technion.ac.il/~gershon/papers/offset-compare.pdf (visited on 04/26/2020).

[17] *Control points of offset bezier curve*. 2019. URL: https://math.stackexchange.com/questions/465782/control-points-of-offset-bezier-curve/467038 (visited on 04/29/2020).

[18] Wonjoon Cho Nicholas M. Patrikalakis Takashi Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing: Approximations*. 2009. URL: http://web.mit.edu/hyperbook/Patrikalakis-Maekawa-Cho/mathe.html (visited on 04/29/2020).

[19] Gabriel Suchwolski. *Quadratic bezier variable offsetting with selective subdivisions*. 2013. URL: https://microbians.com/mathcode (visited on 04/29/2020).

[20] *ISO Double Lane Change Test*. 2020. URL: https://www.vehico.com/index.php/en/applications/iso-lane-change-test (visited on 05/14/2020).

[21] Anton Sergeev. *User Interfaces Design*. 2010. URL: http://ui-designer.net/interface_design.htm (visited on 05/15/2020).

[22] Edwin Hutchins, James Hollan, and Donald Norman. "Direct Manipulation Interfaces". In: *Human-computer Interaction* 1 (Dec. 1985), pp. 311–338. DOI: 10.1207/s15327051hci0104_2.

[23] Anshul vyas. *MVC Pattern*. 2018. URL: https://medium.com/@anshul.vyas380/mvc-pattern-3b5366e60ce4 (visited on 05/17/2020).

[24] *Retopoflow - Retopology Tools For Blender*. 2020. URL: https://blendermarket.com/products/retopoflow (visited on 05/17/2020).

[25] *About Qt*. 2020. URL: https://wiki.qt.io/About_Qt (visited on 05/17/2020).

[26] *Aintree*. 2020. URL: https://www.racingcircuits.info/europe/united-kingdom/aintree.html#.XsTy7Gj7SUk (visited on 05/19/2020).

[27] *Aspen Racing And Sports Car Club*. 2013. URL: http://trackpedia.racetrackdriving.com/wiki/Aspen_Racing_and_Sports_Car_Club (visited on 05/19/2020).