Generating Race Tracks With Repulsive Curves

1st Lasse Henrich
Baden-Württemberg Cooperative State University, Germany
Stuttgart, Germany
lasse.finn.henrich@gmail.com

2nd Falko Kötter
Baden-Württemberg Cooperative State University, Germany
Stuttgart, Germany
koetter@dhbw-stuttgart.de

2024 IEEE Conference on Games (CoG) | 979-8-3503-5067-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/CoG60054.2024.10645670

***Abstract*—Procedural content generation is a useful tool for content creation in computer games. We present a novel approach for generating closed, arcade-like race tracks. Our system generates an initial shape by growing a curve inside a constrained space under self-repulsion, hence avoiding self-intersections while ensuring tight packing. This system is capable of generating a wide variety of closed race track shapes with deep customization options, like an isometric alignment of straight sections as developed here. Further, we devise algorithms for fitting the shape to a spline and introducing intersections, as well as fully generating a 3D model with features like crossings and bridges. We integrate the generated tracks into a game by training AI opponents and using them for automatic difficulty evaluation.**

***Index Terms*—procedural content generation, racing games, race track generation, racing AI, difficulty evaluation**

I. Motivation

As video game revenue shifts from up-front sales to microtransactions, player retention becomes a key challenge to a game's financial success [1]. Live service games focus on continuous content delivery and thus revenue from an existing player base [2], while free-to-play games aim to retain players to make or keep them spending [3].

Algorithms and tools to (semi)automate content creation in video games are called *Procedural Content Generation (PCG)* [4]. PCG can happen on-the-fly while games are played or be used as an authoring tool during game development. For live service games, it can ease the cost of player retention by making regular content updates less expensive. However, content updates only increase player engagement if they're of high quality [5].

In this work, we present a PCG algorithm for creating high-quality race tracks for the chaotic fun-racer *Pocket Racer*, which features elements like crossings, ramps and bridges. As an indie game developed by a solo developer with the *Unity Engine*, content creation is especially burdensome, thus necessitating automation. The resulting algorithm will be used in an authoring tool, so manual curation and tweaking of race tracks is possible.

As will become clear in the following section, current approaches generate curves of usually undesirable quality, mostly caused by unwanted tight spots or overlaps. Therefore, our work focuses on creating tightly packed curves with only a few, carefully placed self-intersections.

II. Related Work

In this section we review relevant work for race track generation in gaming.

[6] aims to increase player satisfaction in racing games using artificial intelligence. Togelius et al. define five criteria for fun racing games: Sensation of speed, limiting straight track segments, level and variety of challenge, as well as drifting in turns. They generate race tracks with an evolutionary search-based algorithm, starting with a set of straight segments, which are mutated throughout the generations into obstacles or curves. Via vehicle simulations the fitness is calculated regarding the expected progress, process variation, and maximum speed. The authors deem the results as satisfactory, generating human-drivable and well-designed tracks. However, in comparison to this work, the approach is limited as tracks are jagged and don't form a continuous circuit.

In [7], Togelius et al. extend their algorithm to address these limits, thus creating smoother, looping tracks using a Bézier spline with 30 segments. Evolution occurs by modifying the control points of these segments. It is tailored to specific players, whose preferences are emulated using player models trained with AI. While the authors find the new approach overall an improvement, they argue that the previous, segment-based method is more expressive. Additionally, crossings are not possible.

Similarly, [8] personalize race tracks based on an individual's driving style. After a recorded race, an existing track is adjusted for the next game by modifying segments as splines and adjusting the track's 3D model. The authors assume that difficulty is raised by increasing a turn's angle or the number of turns.

Loiacono et al. apply the approach of Togelius et al. to The Open Racing Car Simulator (TORCS). Their goal is a large degree of track diversity, and they change the fitness function to encompass a curvature profile and a speed profile [9]. This leads to circular tracks that are rated by human observers. The survey finds that the updated fitness function matches the visual preference of participants and that the perceived difficulty correlates to diversity within a track.

[10] present another search-based approach in which tracks are also generated via an evolutionary algorithm that uses speed and curvature profiles as a fitness function. The authors generate circuits, but explicitly exclude crossings. Compared to other approaches, generated tracks show a clear demarcation

TABLE I
COMPARISON OF APPROACHES FOR TRACK GENERATION IN RACING GAMES

work	focus	algorithm	track type	output	crossings	smoothing	height	scenery	sitsp
[6]	5 fun metrics	search-based (evolutionary)	linear	2D track	○	○	○	○	n/a
[7]	personalization	search-based (evolutionary)	circuit	2D track	○	●	○	○	○
[8]	difficulty modification	adaptive	circuit	rFactor 2 (3D)	○	●	○	○	◑
[9]	track diversity	search-based (genetic)	circuit	TORCS (3D)	○	●	○	○	○
[10]	fun metric	search-based (genetic), tabu search	circuit	TORCS (3D)	○	●	○	○	◑
[11]	user feedback	search-based (interactive genetic)	circuit	TORCS (3D)	○	●	●	●	○
[12]	isometric 2D racing	constructive using chain codes	circuit	2D spritemap	●	●	○	○	◑

between curves and longer straight parts. Once again, evaluation by humans is favorable.

[11] uses real user feedback as a fitness function for track generation. Tracks are available for players on a download server and are evaluated after gameplay. Notably, a finished track is automatically generated, including a height profile and scenery objects.

In [12] Nascimento et al. develop a method for PCG of isometric race tracks using chain codes. Chain codes describe objects by a series of boundary coordinates, in this case race tracks. Existing race tracks are converted from image files to chain codes, and then procedurally recombined to create new ones. Results are smoothened into a polyline with fewer points with the Ramer-Douglas-Peucker algorithm, similar to the curve fitting step in this work. While this approach creates high-quality results, preparation work is high compared to other approaches and variability is limited. A prototype in Unity is evaluated using volunteer players regarding the aforementioned criteria defined in [6].

Table I shows a comparison of reviewed works in procedural race track generation. While previous approaches can generate circuits, none is refined enough to tackle the problem of unwanted self-intersections in tight spots (sitsp). With these approaches, it may happen that two segments lie too close to each other or that some curves are too sharp, resulting in unnatural 3D models that only vaguely resemble race tracks. This work presents a novel approach based on unknotting to create varied circuits with crossings and solve this problem.

III. GENERATION PROCESS

The generation process is divided into five steps, as pictured in Figure 1. We begin with the generation of the two-dimensional shape of the curve by first growing an initially circular polyline under self-repulsion and constraints, followed by a separate refinement of this shape to conform with isometry. This shape generation is the central topic of our work. In the third step, we use a curve fitting algorithm to construct a spline of cubic Bézier curves from the polyline. After having introduced intersections in the fourth step, we finally generate the 3D model of the race track.

A. Shape Generation

To solve the problem of unwanted self-intersections within the model, generating closely packed curves with minimal tight spots is the base goal of our algorithm.

1) Self-repulsion: Our solution applies a previously unconnected field of study, the *unknotting problem*: Can a knot—a closed curve—be transformed into a circle without passing through itself? The process of unknotting is especially relevant for this work. Unknotting algorithms try to minimize a knot's *energy*, which is calculated based on its current shape. Tight spots generally result in higher energies and are therefore penalized, while self-avoidance is rewarded.

For discussing knot energies and their minimization, we use $\gamma : M \to \mathbb{R}^2$ to represent a parametric curve. The norm of values and the power of sets are denoted by $|\cdot|$, while vector norms use $\|\cdot\|$. We define the cross product $a \times b$ of two vectors $a, b \in \mathbb{R}^2$ as $a_1 b_2 + a_2 b_1$. An energy $\mathcal{E}$ of a curve γ is given by

$$\mathcal{E}(\gamma) = \iint_{M^2} k(x, y)\, dx_\gamma\, dy_\gamma, \tag{1}$$

where $(x, y) \in M^2 := M \times M$. The kernel $k : M^2 \to \mathbb{R}$ describes the interaction of two points along the curve.

In *Repulsive Curves* [13], Yu, Schumacher & Crane develop an algorithm that can efficiently untangle knots into shapes with the least possible energy, whereas previous solutions focus on the resolution of local tight spots.

The energy used in [13] is the tangent point energy (TPE) after [14] and [15], defined by the kernel

$$k_{\mathrm{TPE}}(x, y) := \frac{1}{r(x, y)^\alpha}, \tag{2}$$

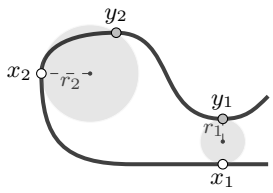

Fig. 2. Tangent point energy

where $r(x, y)$ denotes the radius of the smallest circle that is tangential to $\gamma(x)$ and that passes through $\gamma(y)$. The energy is inversely related to the distance between points $\gamma(x)$ and $\gamma(y)$, approaching infinity for spatially proximate but curve-distant points, and decreasing for points close along the curve.

More explicitly, the circle's radius between two points $\gamma(x)$ and $\gamma(y)$ can be described as

$$r(x, y) = \frac{\|\gamma(x) - \gamma(y)\|^2}{|T(x) \times (\gamma(x) - \gamma(y))|}, \tag{3}$$

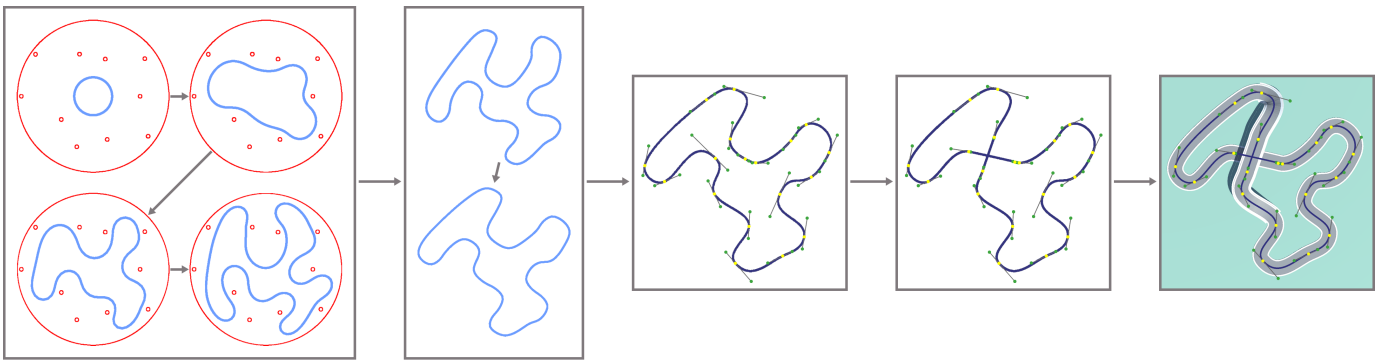

Fig. 1. Main steps of the race track generation

with $T(x)$ being the unit tangent of γ at x. Inserted into (1) and (2) and generalized as shown in [16], Yu et al. describe the TPE as

$$\mathcal{E}_{\alpha,\beta}(\gamma) \coloneqq \iint_{M^2} k_{\alpha,\beta}(\gamma(x),\gamma(y),T(x))\,dx_\gamma\,dy_\gamma \tag{4}$$

with the tangent point kernel

$$k_{\alpha,\beta}(p,q,T) \coloneqq \frac{|T\times(p-q)|^\alpha}{\|p-q\|^\beta}. \tag{5}$$

The parameters α and β control the bending behavior. Here, we set $\alpha = 3$, $\beta = 6$ for all figures, resulting in strong repulsive forces near tight spots while keeping the formula well-defined. As evaluated in [13], the TPE offers significant advantages over other commonly used energies for computational design, making it our preferred choice.

To remove tight spots in a curve, we represent it as a closed polyline, calculate the energy between all pairs of points and minimize the total energy by moving the points using gradient descent. Given appropriate preconditioning, this can transform the curve into a shape with minimal energy.

To calculate the energy between pairs of points, we first need to discretize it. Similar to [13], we describe the knots and edges of the polyline as a graph $G = (V, E)$ with coordinates $\hat{\gamma} : V \to \mathbb{R}^2$. Since the TPE is infinite for polylines [17, Fig. 2.2], we assume $\hat{\gamma}$ to be a smooth curve through V. Each edge $I \in E$ is defined as the set of its vertices i_1 and i_2. The length of I is $l_I \coloneqq \|\hat{\gamma}_{i_1} - \hat{\gamma}_{i_2}\|$, its unit tangent is $T_I \coloneqq (\hat{\gamma}_{i_2} - \hat{\gamma}_{i_1})/l_I$, and $m_I \coloneqq (\hat{\gamma}_{i_1} + \hat{\gamma}_{i_2})/2$ describes its midpoint.

Following *Repulsive Curves*, we approximate the discrete energy as

$$\hat{\mathcal{E}}_{\alpha,\beta}(\hat{\gamma}) \coloneqq \sum_{I,J\in E, I\cap J=\emptyset}\sum \hat{k}_{\alpha,\beta}(I,J)\,l_I\,l_J \tag{6}$$

with the discrete kernel

$$\hat{k}_{\alpha,\beta}(I,J) \coloneqq \frac{1}{4}\sum_{i\in I}\sum_{j\in J} k_{\alpha,\beta}(\hat{\gamma}_i,\hat{\gamma}_j,T_I). \tag{7}$$

To calculate the energy between pairs of points instead of edges, we further approximate the energy between I and J by only considering m_I and m_J. For not having to calculate and move midpoints and instead being able to work with V directly, we assume that V describes the midpoints of imagined edges of an undefined polyline. Thus, the discrete energy based on V is given by

$$\hat{\mathcal{E}}^{\mathcal{V}}_{\alpha,\beta}(\hat{\gamma}) \coloneqq \sum_{i,j\in V, i\neq j}\sum k_{\alpha,\beta}(\hat{\gamma}_i,\hat{\gamma}_j,T_i)\,l_i\,l_j. \tag{8}$$

The unit tangent T_i of an imagined edge with $i \in V$ and $I_1, I_2 \in E, I_1 \cap I_2 = \{i\}$ is defined as $T_i \coloneqq (T_{I_1} + T_{I_2})/\|T_{I_1} + T_{I_2}\|$. The length of the imagined edge is $l_i \coloneqq (l_{I_1} + l_{I_2})/2$.[1]

In our first approach, we tested the energy minimization by generating random curves and applying standard, unpreconditioned gradient descent. The initial curves were produced by selecting random anchor points for a spline of cubic Bézier curves, determining the other control points under the constraint of G^1 continuity, and discretizing that smooth curve to a polyline.

For the gradient descent, we used the discrete differential

$$d\hat{\mathcal{E}}^{\mathcal{V}}_{\alpha,\beta}(\hat{\gamma}) = \begin{bmatrix}\frac{\partial\hat{\mathcal{E}}^{\mathcal{V}}_{\alpha,\beta}}{\partial\gamma_1} & \cdots & \frac{\partial\hat{\mathcal{E}}^{\mathcal{V}}_{\alpha,\beta}}{\partial\gamma_{|V|}}\end{bmatrix} \in \mathbb{R}^{2|V|} \tag{9}$$

as outlined in [13]. This differential determines the descent direction for each $i \in V$, which we define as

$$\frac{\partial\hat{\mathcal{E}}^{\mathcal{V}}_{\alpha,\beta}}{\partial\gamma_i} \coloneqq \nabla\Big(\sum_{j\in V, j\neq i} k_{\alpha,\beta}(\hat{\gamma}_i,\hat{\gamma}_j,T_i)\,l_i\,l_j\Big) \in \mathbb{R}^2. \tag{10}$$

Exemplary screenshots of curves resulting from this approach are shown in Figure 3. As is clearly visible, while the method can reduce tightness in some places, its effectiveness is constrained by the curve's initial shape.

2) Restricted Growth: The initial approach of generating a shape *first* and addressing tight spots *after* appears ineffective. Instead, the shape generation process should be guided by energy minimization. Yu et al.'s algorithm can be used for *curve packing*, i.e. the generation of closely packed curves with minimal tight spots in a constrained space [13, Section

[1]This approximation is in accordance with an implementation of *Repulsive Curves* by one of its authors [18].

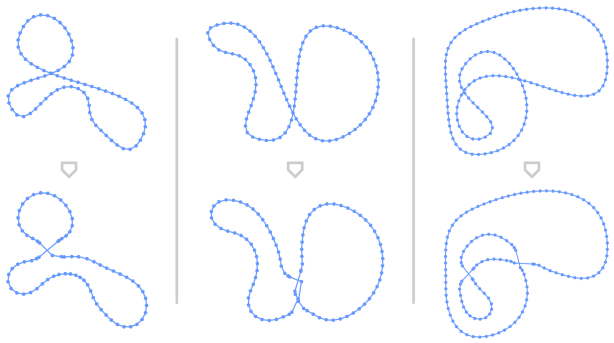

Fig. 3. Curves generated through random Bézier splines and standard energy minimization

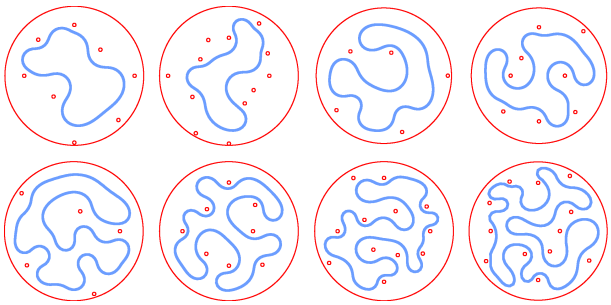

Fig. 4. Curves generated through constrained growth

8.2]. This method starts with a simple shape, e.g. a circle, and forces it to expand while applying the gradient descent for energy minimization simultaneously.

To enable growth in reasonably big steps, the existing ordinary gradient descent must be replaced with a specialized fractional Sobolev descent. This process is mathematically complex and forms the essence of *Repulsive Curves* (ref. [13, Section 4.2 & Appendix A]. In principle, a well-matched Sobolev descent can address both local and global features at the same time. When applied to our previously generated curves, this method not only eliminates local tight spots but further transforms the curves into ever-expanding circles or similar low-energy shapes.

For encouraging growth within a restricted space despite potential energy increases, we implement the constraint and potential system from *Repulsive Curves*. This system further allows us to meet arbitrary design requirements, such as an isometric alignment of straight sections.

Optimization under constraints is handled using Lagrange multipliers [19]. Each constraint i is represented by $\Phi_i(\gamma) = 0$. The Sobolev gradient g is first projected onto a valid descent direction $\tilde{g}$, aligning as closely as possible with the original direction while adhering to the constraints. Then, the curve candidate itself is projected back onto the constraint set.

Potentials are a less rigid form of constraints, influencing the curve similarly to how the curve influences itself. For each potential p, we define a function $\Psi_p(\gamma)$ describing its energetic impact. To reduce the exerted energy, we must find $\nabla\Psi_p : E \to \mathbb{R}^4$, the energy gradient for each edge's vertices.

As detailed in [13, Section 8.1], we induce curve growth by setting a constraint $\Phi_{length,I}(\gamma) \coloneqq l^0 - l_I$ for each edge $I \in E$, with l^0 representing the target edge length for a given step. Similar to [18], we ensure that each edge is bisected once it doubles in length.

To facilitate curve shaping, we create spatial obstacles in the form of potentials. Starting with an initially circular polyline of radius r_γ, we limit space using a larger outer circle of radius r_o and multiple randomly positioned inner obstacles of radius r_i. These inner obstacles are positioned based on a distance $d_i \in (r_\gamma + r_i, r_o]$ from the center and an angle $\alpha_i \in [0, 2\pi)$.

Figure 4 shows some typical examples of curves generated using this approach, varying in total length and number of inner obstacles. The effectiveness of our approach is apparent.

3) Isometry: In Pocket Racer, race tracks are optimized for a mobile device's format through isometrically aligned straight sections. To achieve the same effect for generated curves, we can leverage the previously defined potentials. Following [13, Section 8.1], we apply a potential $\mathcal{E}_X(\gamma)$ for a vector field X on $\mathbb{R}^2$. It's discrete form is given by

$$\hat{\mathcal{E}}_X(\gamma) \coloneqq \sum_{I \in E} l_I |T_I \times X(I)|^2. \tag{11}$$

This energy decreases as the curve γ aligns more closely with X. It's gradient can be computed based on the following expansion:

$$\begin{aligned} &\nabla(l_I |T_I \times X(I)|^2) \\ =&(\nabla l_I)|T_I \times X(I)|^2 + l_I 2|T_I \times X(I)| \\ &\cdot (\nabla T_I \times X(I) + T_I \times \nabla X(I)) \\ =&(\nabla l_I)|T_I \times X(I)|^2 + l_I 2|T_I \times X(I)| \\ &\cdot (\nabla T_I \times X(I)) \quad \text{when} \quad \nabla X(I) = 0. \end{aligned} \tag{12}$$

This calculation assumes that the vector field X is constant along the curve, i.e. $\nabla X(I) = 0$.

For the desired isometry, we define

$$X_{\text{Iso}}(I) \coloneqq \begin{cases} \begin{bmatrix} 1 & 1 \end{bmatrix}, & \text{if } \arctan(T_{I,y}/T_{I,x}) \geq 0 \\ \begin{bmatrix} -1 & 1 \end{bmatrix}, & \text{otherwise,} \end{cases} \tag{13}$$

with $\nabla X_{\text{Iso}} = 0$, as it depends only on the edge's tangent, not its position.

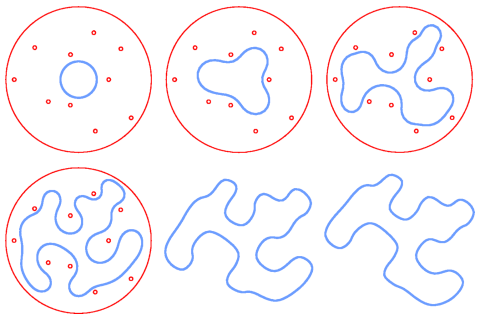

Fig. 5. Curve generation with subsequent isometry potential

We initially generate the shape of the curve as described, then disable all obstacles and apply the isometry potential afterwards, as an application of the potential from the start forces the curve into an unnatural shape. When applying the potential, we maintain the TPE to avoid creating sharp corners.

While this method produces aesthetically pleasing results, applying the potential in the second phase results in some information loss from the initial curve. Increasing the weight of the isometry w_{Iso} can mitigate this, though it also tightens some turns. Figure 5 showcases race tracks generated with the same initial configuration but varying weights. For all curves presented in this paper, we use a weight of $w_{\text{Iso}} = 0.1$.

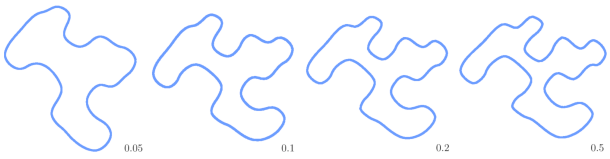

Fig. 6. Effect of different isometry potential weights

B. Curve fitting

Next we convert the discrete polyline into a continuous spline, enabling re-discretization into finer segments as needed for various applications. Additionally, a spline offers easier editability compared to a polyline.

One easily editable spline is one of Bézier curves with G^1 continuity. The method presented in [20] for fitting a polyline to such a spline can effectively be applied here.

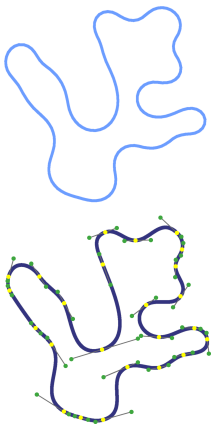

Fig. 7. Curve fitting

The algorithm works recursively: It starts by trying to represent the entire polyline with a single cubic Bézier curve. If the error, calculated as the sum of squared distances from the polyline's points to the curve, exceeds a threshold $\epsilon \in \mathbb{R}$, the two control points of the Bézier curve are adjusted to reduce it. For most polylines, a single Bézier curve isn't sufficient, so if the error is above a second threshold $\psi \in \mathbb{R} > \epsilon$, the set of points is split and the algorithm is reapplied to each subset. Finally, the resulting Bézier curves are combined into one spline.

In our work, we set $\epsilon = 0.2$ and $\psi = 2$. This choice ensures a fast and precise approximation of the polyline, retaining most of its information, while avoiding an excessive number of control points in order to preserve editability. An exemplary result of this curve fitting algorithm is shown in Figure 7.

C. Introducing intersections

Most tracks of Pocket Racer—as well as of some other fun racers—feature elements like crossings or bridges over parts of the track. Therefore, we use this section to devise a short algorithm for introducing intersections into previously generated Bézier splines.

We decide against using overlapping curves from the outset, as constrained growth doesn't treat intersections naturally (see Figure 3). Direct manipulation of energy near intersections is more complex compared to the subsequently presented approach and might negatively impact the variety of curve shapes.

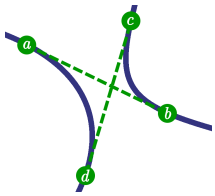

Fig. 8. Concept for introducing intersections

Our method for introducing intersections is straightforward and can be summarized in three steps:

1) We identify all point pairs $\{a, b\}$ that can be connected by a relatively straight line segment $[ab]$.
2) We evaluate the quality of all spline candidates formed by connecting two such straight segments $\{[ab], [cd]\}$.
3) The highest-quality candidate becomes the new spline.

To iterate through points on the spline, we first discretize it back to a polyline. For cubic Bézier curves, this process is detailed in [21, §27]. We adapt this algorithm for splines of Bézier curves, allowing us to generate closely spaced points.

In the second step, a spline candidate is created by opening up the original spline at points a, b, c, d and reassembling it in the correct sequence. For that, new anchor points are initially generated at a, b, c, d, which we realize by dividing the spline segment s_i of a point $x \in \{a, b, c, d\}$ at t, where $s_i(t) = x$. To retain the curve's shape during the division, we use the algorithm showcased in [21, §10].

Merging the spline back together is complex due to the varying order in which a, b, c, d should appear along the new spline. Due to the number of special cases, we won't describe the implementation in more detail here.

For step 3, the quality of each candidate is assessed: In order to avoid excessively long straights, the quality increases the shorter the newly introduced straight sections are. Secondly, the quality decreases the more the candidate's length deviates from the original spline's length, helping in retaining most of the original characteristics. Thirdly, right angles between the two straight sections are rewarded and acute and obtuse angles are penalized indefinitely. These criteria balance the introduction of intersections with preserving the spline's original shape.

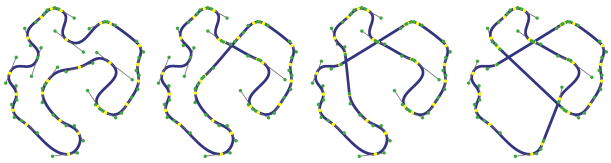

Fig. 9. Introducing three intersections

Figure 9 illustrates the process of applying the described algorithm three times. After the third application, no more pairs $\{a, b\}$ are found during the first step, preventing the introduction of further intersections.

Applying the algorithm as often as possible generally yields favorable results. Moreover, our approach allows for the integration of user-defined quality metrics. However, some splines don't pass the first step and thus remain without intersections. Furthermore, without a minimal quality threshold for candidate selection, introducing intersections sometimes results in losing significant parts of the track.

D. 3D model creation

To ensure different kinds of sections, we develop a simple function $f(x) : M \rightarrow F$ that assigns a feature $F \in \{\mathrm{Road}, \mathrm{Crossing}, \mathrm{Bridge}, \mathrm{Ramp}\}$ to each point on the curve. Our requirements for this function are that the length of a feature is independent of the constellation of spline segments, and that certain features like Bridge maintain a constant length, unaffected by the total length of the spline.

The first step in placing these features is to locate all intersections along the spline. For this, we use the intersection detection algorithm described in [21, §29], modifying it to return the position of each intersection.

Our initial goal is to create correct race tracks, with the possibility of future enhancements to assess the quality of feature placements. The algorithm processes each intersection, evaluating for each feature $F \neq \mathrm{Road}$ whether it can be placed around it. Then, a suitable feature is selected and added. For variable-length features like Bridge, the maximum and minimum lengths are determined, and a value within this range is randomly selected.

A simple method for generating a 3D model along a spline is to extrude a pre-defined edge loop [22]. This method, however, is not suitable here as it limits the incorporation of some features and more complex models. While advanced tools for spline-based modeling exist [23] [24], they rely on user input and are not designed to generate entire roads by themselves. Instead, we allow developers to specify a 3D model for each feature and procedurally replicate it along the spline.

Creating a 3D crossing model is challenging as it requires simultaneous consideration of both intersecting spline segments. While some publications address this issue [25, Section 3.6] [26, Section 4.2.2], none offer a solution that allows for at least some artistic freedom.

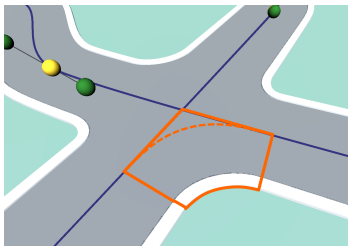

Fig. 10. Crossing from reference model

Mapping a predefined crossing model to an intersection of curved segments is difficult, and we have not found a transformation capable of doing this. Instead, we developed a solution as illustrated in Figure 10. Here, an artist designs only *one fourth* of the crossing, which is then used to connect all halves of neighboring road ends. We use Bézier curves to connect each such pair, leveraging the same functions employed for standard road generation. Finally, the central gap is closed by adding a vertex at the intersection and forming faces with the inner edges of the outer segments.

To ensure that the generated 3D model is tightly packed but extensive enough to avoid unwanted self-intersections, a method for determining the optimal scale of the initial curve is required. Our approach first finds the minimum road width $\hat{w} \in \mathbb{R}$ that prevents self-intersections in the 3D model for the unscaled curve. Once $\hat{w}$ is identified, we scale the curve by $\frac{1}{\hat{w}}$ prior to initiating the model generation process.

We consider a width w too great if two normals of length $\frac{w}{2}$ intersect. Although this assumption doesn't cover the case of two straight, parallel segments, it works in practice due to adjacent curves and a sufficient number of evaluated normals along the spline. Furthermore, this assumption allows to find $\hat{w}$ by simply calculating the intersections of all pairs of normals, determining how far along the normals the intersection occurs, and using twice the smallest of all values as $\hat{w}$.

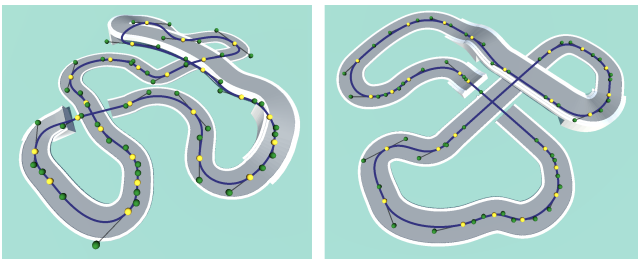

Fig. 11. Exemplary generated 3D models
Both splines were edited after the constrained growth.

IV. Game Integration

How well our approach can be integrated into games depends on various factors, which we explore and develop solutions for in this section.

A. Benefits of using Bézier splines

Choosing cubic Bézier splines significantly enhances editability. Many spline editing tools compatible with game engines like Unity are readily available, and we integrated one here as well. Additionally, track information can be efficiently stored as positions of the spline's anchor and control points along with the feature list, minimizing storage requirements.

In contrast to traditionally designed race tracks, our system offers an easy-to-use mathematical representation of the curve, enabling more gameplay features. By dividing the spline into lots of points, we can continually evaluate which of those points the vehicle is closest to. Without much further effort, this allows for determining vehicle placements. It also helps in resetting a vehicle to the track's center, facing forward, if it significantly deviates from the course, such as when cutting corners or reversing.

B. Opponent AI

To enable game developers to directly use generated maps in their games, we must also develop a solution for AI-controlled vehicles to drive on the tracks. As developing such bots is not

the central research question of this publication and multiple reports on this topic are already available [27] [28] [29] [30], we focus only on details specific to our use case.

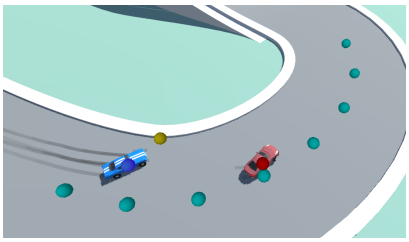

Fig. 12. Visualization of the AI model's spatial inputs & outputs
Green spheres mark the track, red spheres indicate other vehicles' positions, and the yellow sphere shows the inferred direction, all relative to the blue sphere.

Similar to most modern approaches, we use Reinforcement Learning (RL) to train neural networks, as mathematically describing the entire game world for trajectory planning on a continually adapted route is too complex. An ML-based approach allows us to combine route finding and trajectory planning computationally. Moreover, we leverage the framework *ML-Agents* [31], simplifying RL integration significantly.

We train the AI on multiple race tracks simultaneously to include a variety of road features and mitigate overfitting to one track. Additionally, for each episode tracks are rotated and vehicles placed in different positions. To account for diverse vehicle types, six vehicle-physics parameters—like maximum velocity or different kinds of drag—are varied per episode.

Regarding input parameters, the car's state is represented by its rotation and velocity. For the tracks, equally spaced points along the spline are calculated initially, with every tenth point used as input, starting from seven points behind the car up to the 63rd point in front of it. This prevents large jumps in coordinates when the next point is reached. Importantly, we found that using the points' positions relative to the vehicle's position results in much faster training than using absolute coordinates. This also eliminates the need for the vehicle's position as separate input parameters. We further discovered that multiple vehicles should drive together on a road during training to prevent all vehicles from steering to the center of the track during gameplay. Additional input parameters for the relative positions of other vehicles are added for this.

During training, we use a simple reward function that (i) rewards 5 points whenever the next waypoint is reached, and (ii) subtracts 2 points during each step the vehicle is outside of the road. Despite its simplicity, this function ensures that vehicles stay on track. If the vehicle wanders too far from the road or stays on track for 1000 steps, the episode is ended.

We use two output parameters for the network, coordinates that represent a position relative to the vehicle. Those coordinates are treated as user input and can therefore directly be applied in our central vehicle controller.

After optimizing hyperparameters through various tests and training for multiple hours, the result is a model that performs well with no vehicles around, but still has a slight tendency to the center of the road.

C. Measuring difficulty

As our track generation approach aims to produce hundreds of tracks for studios with tight budget and time constraints, we cannot rely on real users to play and rate all tracks. Accurately quantifying track difficulty with a mathematical function is complex [32]. Therefore, we opt for letting the AI drive all tracks and measuring certain metrics.

Each race track is driven twelve times with different, predefined configurations. These configurations vary in variables such as the number of opponents, starting grid positions, maximum velocity limits, and various aspects of rubber-banding dynamics, capturing a wide range of game scenarios.

During each run, we record three performance indicators for the vehicle representing the player: (i) its final position pos, (ii) the number of resets necessitated by off-track excursions res, and (iii) the average time required to cover a unit distance time. The track's length is incorporated as a fourth metric len.

A weighted sum model allows us to assign different weights to those metrics. First, we normalize each metric k to a scale of $[0, 1]$, represented by S_k. The difficulty $\mathcal{D}$ is then given by

$$\mathcal{D} = 1S_{\mathrm{pos}} + 5S_{\mathrm{res}} + 3S_{\mathrm{time}} + 2S_{\mathrm{len}}. \tag{14}$$

The number of resets res is the most important factor, as resets can frustrate the player and therefore make the track feel difficult (ref. [33, Section IV.B]). The time metric typically increases with the number of tight turns, an indicator deemed crucial by other researchers as well (ref. [8, p. 5]). Togelius et al. use a similar metric in a related context [7, p. 256-257]. A great length len can make a track harder to memorize, but is not a defining metric. pos is easily influenced by randomness and mostly already accounted for within the other metrics.

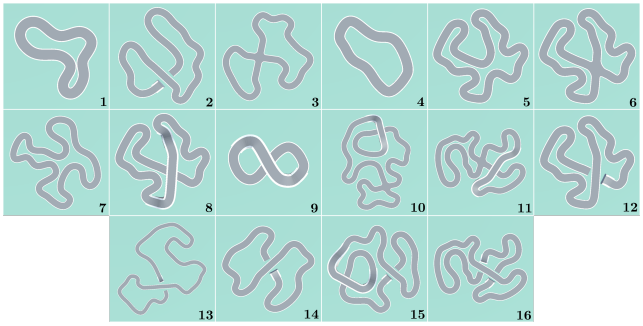

Fig. 13. Generated tracks ordered by increasing difficulty
Tracks scaled for representation purposes—all of them have the same width. Not all tracks are optimized for isometry.

Looking at Figure 13, it's clear that ramps significantly increase difficulty. This is reasonable, as players might fail ramp jumps if they're too slow or don't jump straight, especially after sharp turns. Conversely, crossings and bridges have minimal impact on difficulty. By design, longer tracks are considered more difficult.

Automatic difficulty assessment is complex due to subjective preference and hard-to-define parameters. [34] show that the perceived challenge in casual platform/racing games rises with additional obstacles, even if they don't change the actual

path to be taken to win. Our AI-based assessment works well overall but cannot account for these human perceptions and creativity, such as finding shortcuts or jumping tracks. As our concept primarily serves as an authoring tool, developers can use this for quickly categorizing tracks with the possibility for later adjustments.

V. Conclusion

Applying the unknotting problem to race track generation is a promising approach. We presented an algorithm that generates tightly packed, smooth race tracks with intersections while respecting constraints like isometry. We have shown how this algorithm supports developers by preserving editability, generating 3D models, training AI and estimating difficulty.

Our implementation is still a proof of concept. The algorithms for introducing intersections and generating 3D models with special sections should be replaced by more sophisticated ones. Generating longer tracks can take several minutes on a standard PC, so performance improvements are necessary.

Currently, the presented approach can generate and categorize a variety of tracks for developers to modify and build upon. A generation on the end user's device and presentation without human quality control is not yet in scope.

In the future, we aim to develop a variant of the curve generation algorithm for linear track types. We also plan to incorporate scenery and transition from (essentially) flat tracks to true 3D tracks with height differences. Our approach allows to effectively combine scenery and curve generation by growing the curve directly on the terrain, as illustrated in [13, Fig. 24]. This enables multiple use cases, such as considering the terrain's height profile during curve generation or finding a race track around pre-existing scenic objects.

References

[1] M. Viljanen, A. Airola, A.-M. Majanoja, J. Heikkonen, and T. Pahikkala, "Measuring player retention and monetization using the mean cumulative function," *IEEE Transactions on Games*, vol. 12, no. 1, 2020.

[2] L.-E. Dubois and J. Weststar, "Games-as-a-service: Conflicted identities on the new front-line of video game development," *New Media & Society*, vol. 24, no. 10, pp. 2332–2353, 2022, pMID: 36052017.

[3] N. Hanner and R. Zarnekow, "Purchasing behavior in free to play games: Concepts and empirical validation," in *2015 48th Hawaii International Conference on System Sciences*, 2015, pp. 3326–3335.

[4] N. Shaker, J. Togelius, and M. Nelson, *Procedural Content Generation in Games*. Springer International Publishing Switzerland, 2016.

[5] X. Zhong and J. Xu, "Measuring the effect of game updates on player engagement: A cue from dota2," *Entertainment Computing*, vol. 43, p. 100506, 2022.

[6] J. Togelius, R. De Nardi, and S. M. Lucas, "Making racing fun through player modeling and track evolution," 2006.

[7] J. Togelius, R. de Nardi, and S. Lucas, "Towards automatic personalised content creation for racing games," *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games, CIG 2007*, pp. 252 – 259, 2007.

[8] T. Georgiou and Y. Demiris, "Personalised track design in car racing games," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016, pp. 1–8.

[9] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Automatic track generation for high-end racing games using evolutionary computation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 245 – 259, 2011.

[10] H. Adi Prasetya and N. Maulidevi, "Search -based procedural content generation for race tracks in video games," *International Journal on Electrical Engineering and Informatics*, vol. 8, 2016.

[11] L. Cardamone, P. L. Lanzi, and D. Loiacono, "Trackgen: An interactive track generator for torcs and speed-dreams," *Applied Soft Computing*, vol. 28, pp. 550–558, 2015.

[12] E. J. F. do Nascimento, T. M. Castro, A. C. S. Abreu, F. A. Lira, and A. H. Souza, "Procedural generation of isometric racetracks using chain code for racing games," in *2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2021, pp. 136–143.

[13] C. Yu, H. Schumacher, and K. Crane, "Repulsive curves," *ACM Transactions on Graphics*, vol. 40, no. 2, 05 2021.

[14] G. Buck and J. Orloff, "A simple energy function for knots," *Topology and its Applications*, vol. 61, pp. 205–214, 02 1995.

[15] O. Gonzalez and J. Maddocks, "Global curvature, thickness, and the ideal shapes of knots," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 96, no. 9, pp. 4769–4773, 05 1999.

[16] S. Blatt and P. Reiter, "Regularity theory for tangent-point energies: The non-degenerate sub-critical case," *Advances in Calculus of Variations*, 03 2014.

[17] P. Strzelecki and H. von der Mosel, "On rectifiable curves with l^p-bounds on global curvature: Self-avoidance, regularity, and minimizing knots," *Mathematische Zeitschrift*, vol. 257, pp. 107–130, 09 2007.

[18] C. Yu, H. Schumacher, and K. Crane. (2022-10-21) Github: icethrush/repulsive-curves. [Online]. Available: https://github.com/icethrush/repulsive-curves

[19] D. Bertsekas and W. Rheinboldt, *Constrained Optimization and Lagrange Multiplier Methods*, ser. Computer science and applied mathematics. Elsevier Science, 2014.

[20] P. J. Schneider, "An algorithm for automatically fitting digitized curves," in *Graphics Gems*, A. S. Glassner, Ed. San Diego: Morgan Kaufmann, 1990, pp. 612–626.

[21] Pomax. (2020) A primer on bézier curves. [Online]. Available: https://pomax.github.io/bezierinfo

[22] P. Joshi, "Curve-based shape modeling a tutorial," *IEEE Computer Graphics and Applications*, vol. 31, no. 6, pp. 18–23, 2011.

[23] "The Best Unity Spline Package - Curvy Splines — curvyeditor.com," https://curvyeditor.com/, [Accessed 10-02-2024].

[24] "SplineMesh — Modeling — Unity Asset Store — assetstore.unity.com," https://assetstore.unity.com/packages/tools/modeling/splinemesh-104989, [Accessed 10-02-2024].

[25] J. Freiknecht, "Procedural content generation for games," Ph.D. dissertation, Mannheim, 2021. [Online]. Available: https://madoc.bib.uni-mannheim.de/59000/

[26] X. Zhang, M. Zhong, S. Liu, L. Zheng, and Y. Chen, "Template-based 3d road modeling for generating large-scale virtual road network environment," *ISPRS International Journal of Geo-Information*, vol. 8, no. 9, 2019.

[27] P. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, L. Gilpin, P. Khandelwal, V. Kompella, H. Lin, P. MacAlpine, D. Oller, T. Seno, C. Sherstan, M. Thomure, and H. Kitano, "Outracing champion gran turismo drivers with deep reinforcement learning," *Nature*, vol. 602, pp. 223–228, 02 2022.

[28] C. Hao, C. Tang, E. Bergkvist, C. Weaver, L. Sun, W. Zhan, and M. Tomizuka, "Outracing human racers with model-based autonomous racing," 11 2022.

[29] R. Reiter, J. Hoffmann, J. Boedecker, and M. Diehl, "A hierarchical approach for strategic motion planning in autonomous racing," 12 2022.

[30] R. De Schaetzen and A. Sestini, "Efficient ground vehicle path following in game ai," in *2023 IEEE Conference on Games (CoG)*, 2023, pp. 1–4.

[31] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2020. [Online]. Available: https://arxiv.org/pdf/1809.02627.pdf

[32] R. van der Ploeg, "Modeling race track difficulty in racing games," Master's thesis, Universiteit Utrecht, 2015.

[33] M. A. M. Cuerdo, A. Mahajan, J. Mao, and E. F. Melcer, "Try again?: A macro-level taxonomy of the challenge and failure process in games," in *2023 IEEE Conference on Games (CoG)*, 2023, pp. 1–8.

[34] S. Juyal, "An exploration into "perceived sense of challenge" in level design for fast paced casual mobile games," in *2021 IEEE Conference on Games (CoG)*, 2021, pp. 1–8.