

Bounding Polygon Generation in 2D Games Through Contour Extraction

Maria Laura Barboza Moroni¹, Paulo Henrique Almeida de Andrade¹,
Judson Santos Santiago²

¹Departamento de Computação
Universidade Federal Rural do Semi-Árido (UFERSA) – Mossoró – RN – Brasil

{maria.moroni, paulo.andrade}@alunos.ufersa.edu.br¹,
judson@ufersa.edu.br²

Abstract. Introduction: Collision detection is essential for realistic interactions in 2D games, ranging from pixel-based methods to bounding objects. **Objective:** This research presents an automated tool for generating pixel-perfect bounding polygons from PNG images, aiming for higher precision than traditional AABB and less manual effort than polygon creation. **Methodology or Steps:** The system employs a Radial Sweep algorithm for external contour extraction, ignoring internal holes, and applies a collinearity filter based on Freeman Chain Codes to reduce vertex count, mapping directions between consecutive pixels and discarding intermediate points that maintain the same trajectory. **Results:** The developed tool, AutoBBox, successfully reduced geometric complexity while maintaining pixel-perfect accuracy. In a specific test case involving an irregular character sprite, the solution eliminated the 35% false positive collision area that resulted from standard AABB approximation, and minimized the geometric representation by 39% compared to the vertices typically required in a non-optimized trace.

Keywords Collision Detection, Contour Tracing, 2D Games, Digital Image Processing, Hitboxes

1. Introduction

In the development of two-dimensional games, collision detection is the process of identifying when two or more objects in the game world are touching or intersecting [Lazaridis et al. 2021]. This fundamental aspect enables realistic interactions and ensures that the virtual environment behaves in an intuitive manner for the player.

Game developers generally choose between two primary collision detection systems: pixel-based and bounding object-based [Mileff 2023]. Pixel-based detection offers high precision by examining the overlap of individual pixels within game textures. However, this method is often computationally intensive, and its complexity increases with texture resolution. Bounding object-based detection provides a more efficient alternative by enclosing game elements within simplified shapes such as boxes and performing intersection tests at the level of these enclosing objects.

The most popular bounding object is the bounding box [Mileff 2023], a method where an object is surrounded by a rectangular enclosure, designed to fit as closely as possible to the real object's shape. The most common implementation is the Axis-Aligned Bounding Box (AABB) [Ericson 2004], favored for its mathematical simplicity

and low resource requirements. While AABBs are effective for many scenarios, they are insufficient for objects requiring high-precision boundaries. Specifically, the AABB is insufficient for objects that cannot be accurately represented by a single rectangle, as it is limited to axis-parallel rectangles [Lazaridis et al. 2021] and can lead to false positive collisions, where the box triggers a contact before the visible pixels actually meet.

When greater accuracy is required, a bounding polygon approach is preferred. Bounding polygons are designed to follow the continuous contours of an object more closely than a standard box. Modern game engines support this approach, often providing a separate editing interface for manually creating and modifying the polygon [Linietsky et al. 2014]. While these polygons offer superior precision, they are more complex and can be computationally expensive and slow. Furthermore, the manual creation of these polygons is a time-consuming task for developers.

The objective of this research is to develop an automated tool for generating bounding polygons from PNG image data. The resulting polygon must be pixel-perfect, ensuring that the boundary aligns precisely with the outermost non-transparent pixels of the source texture.

Furthermore, the system is required to output a continuous vertex array, where each point is sequenced to form a closed loop. This structural continuity is essential for compatibility with physics engines and intersection algorithms. Finally, the generation process must focus on the external contour of the object, ignoring internal transparent holes or nested shapes within the texture.

2. Theoretical Framework

The extraction of image components is a digital image processing foundation utilized for extracting "meaning". In the context of binary patterns, where pixels are categorized as either part of a pattern or background, identifying objects requires locating groups of connected pixels [Gonzalez e Woods 2018]. This process is fundamental for the representation of region shapes.

2.0.1. Boundary Representation and Chain Code

Once a contour is extracted, it must be represented in a structured format for computational analysis. Chain codes are used for this purpose, representing the boundary as a sequence of connected straight-line segments of specified length and direction. When a boundary code is structured as a sequence of directional numbers, it is known as a Freeman Chain Code [Freeman 1961]. While 4-connectivity restricts transitions to horizontal and vertical movements between neighboring pixels, 8-connectivity incorporates diagonal movements. Figure 1 illustrates all the possible paths a line can follow starting at point (X, Y) .

2.0.2. Contour Tracing Algorithms

Contour tracing extracts the boundary of digital patterns. Two widely used 8-connectivity methods are Moore-Neighbor Tracing and Radial Sweep. Moore-Neighbor

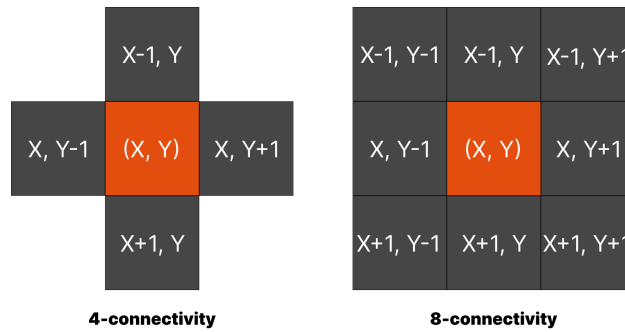


Figure 1. Representation of connected components

[Reddy et al. 2012] searches the 8-pixel neighborhood clockwise from the last visited transparent pixel, but its fixed backtracking can lead to unintended paths in complex topologies. Radial Sweep [Alder 1997] resolves these ambiguities through a directionally aware search, identifying the next boundary pixel by rotating a line segment from the current to the previous pixel clockwise until it intersects the foreground.

3. Related Works

Automated approaches have emerged to address limitations by extracting contours directly from sprite textures, enabling pixel-perfect representations suitable for physics engines.

Thompson [Thompson 2023] presents a script for the Godot editor tool that regenerates CollisionPolygon2D nodes upon sprite texture changes, leveraging the engine's BitMap.opaque_to_polygons method with zero epsilon for pixel-perfect accuracy. This script processes the sprite's alpha channel into multiple polygons, handling disconnected regions automatically, but remains engine-specific and integrated into the editor workflow rather than a standalone generator.

Mandachescu [Mandachescu 2003] proposes precomputed polygonal bounding boxes (four or more edges) for 2D game pathfinding, automatically generated from object shapes, vertex-minimized, and dynamically triangulated for navigation meshes. While effective for obstacle avoidance, the focus lies on path planning rather than real-time collision detection.

4. Methodology

The automatic bounding box creation process was divided into four stages: PNG image importation, searching for the first pixel, application of the Radial Sweep algorithm, and vertex exportation, as shown in Figure 2.

4.1. Border extraction

The border extraction process begins with a scan of the PNG image to identify the initial boundary point. The algorithm performs a row-major traversal, evaluating the alpha channel of each pixel until it encounters the first non-transparent pixel ($\alpha > 0$). This coordinate is defined as the starting index (S) and serves as the reference for the stopping criterion, as shown in Figure 3.

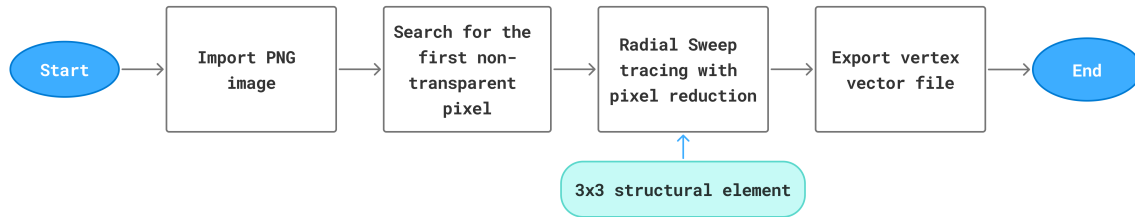


Figure 2. Methodology diagram

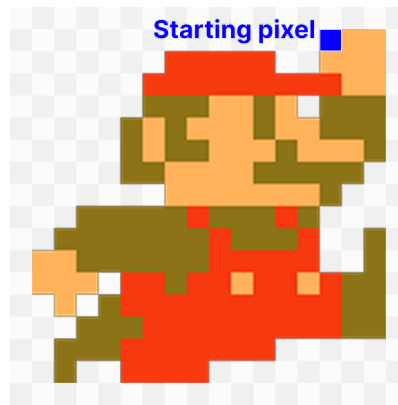


Figure 3. Mario's starting pixel example

Once the starting point is identified, the algorithm employs a Radial Sweep approach using a 3×3 structuring element to evaluate the neighborhood around the current pixel (X, Y) . By strictly initializing the search at the top-leftmost non-transparent pixel, the traversal is anchored to a predictable geometric behavior, eliminating the need for complex stopping criteria.

The search for the subsequent boundary pixel follows a counter-clockwise rotation, starting from a relative offset determined by the direction of the previous movement. This ensures that the algorithm consistently hugs the outer contour of the shape. For each step, shown in Figure 4, the algorithm evaluates the neighbors; the first pixel found with an alpha greater than zero is designated as the new current boundary pixel.

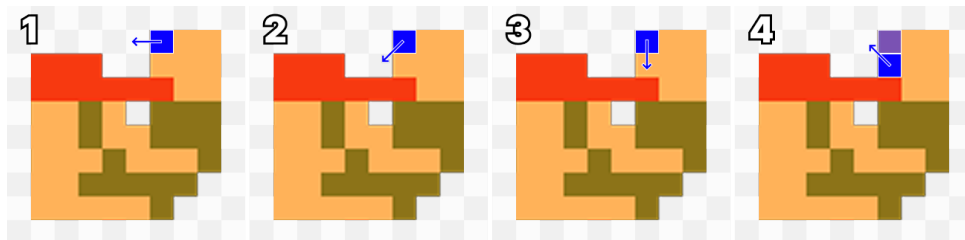


Figure 4. Radial Sweep algorithm

To prevent infinite loops, a dual stopping criterion is implemented: the process terminates when the tracer returns to the starting index S , or when the total number of iterations reaches the total pixel count of the image.

4.2. Vertex vector construction

The raw output of the border extraction is a sequence of every pixel on the external contour. However, storing every pixel as a vertex can be computationally inefficient. To optimize the geometric representation, a new vertex is only appended to the vector if there is a change in the movement direction. If the current pixel continues in the same direction as the last recorded vertex, it is considered part of a straight line segment and is discarded.

5. Results

The developed tool, called AutoBBox¹, was implemented to generate bounding polygons for game projects using the Volt Game Engine [Santiago 2022]. The tool was developed and achieved its goal of reducing geometric complexity while maintaining pixel-perfect accuracy, as shown in Figure 5.

Quantitatively, the limitations of traditional methods are evident when compared to automated extraction. A standard square approximation (AABB) applied to an irregular character sprite results in a false positive area of 35% of the image, where the physics engine registers a collision despite no visible pixels intersecting. To achieve a pixel-perfect boundary without point reduction, the vertex map size would go up to 170 vertices.

For the same example, the AutoBBox solution consolidates pixel-perfect precision with a minimized geometric representation of 104 vertices, providing a vertex reduction of 39% compared to what is typically required in a manual or non-optimized trace, as shown in Figure 5.

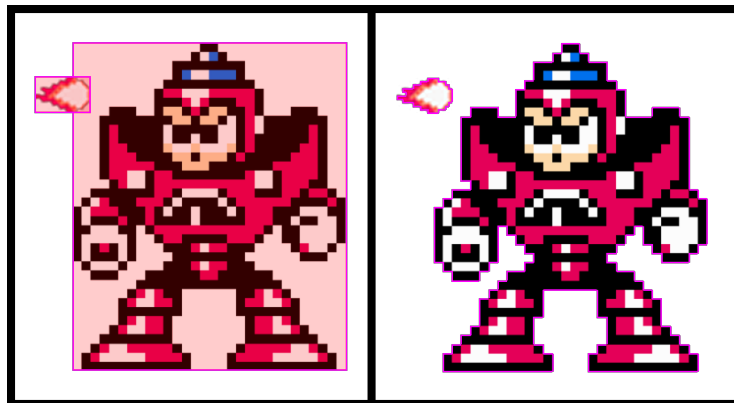


Figure 5. Gravity Man Square Bounding Box vs. Pixel-Perfect Collision

In contrast to engine-bound solutions like Thompson’s Godot tool, our system operates as an independent processor for PNG images, producing a serialized, continuous vertex array focused exclusively on the external contour. Unlike Mandachescu’s pathfinding-oriented polygons, which prioritize triangulation and vertex minimization for navigation, our approach enforces pixel-perfect external boundaries.

6. Conclusion

This work presented AutoBBox, an automated tool designed to generate pixel-perfect bounding polygons for 2D games. The system successfully demonstrated that integrating

¹AutoBBox source code: <https://github.com/lauramoroni/AutoBBox>

the Radial Sweep algorithm with a Freeman Chain Code collinearity filter provides a precise collision boundary, significantly minimizing geometric complexity without sacrificing visual fidelity.

References

- Alder, M. (1997). An introduction to pattern recognition: Statistical, neural net and syntactic methods of getting robots to see and hear. Course notes, Department of Mathematics, University of Western Australia.
- Ericson, C. (2004). *Real-Time Collision Detection*. CRC Press, Boca Raton, FL, USA.
- Freeman, H. (1961). On the encoding of arbitrary geometric configurations. *IRE Transactions on Electronic Computers*, EC-10(2):260–268.
- Gonzalez, R. C. e Woods, R. E. (2018). *Digital Image Processing*. Pearson, 4th edition.
- Lazaridis, L., Papatsimouli, M., Kollias, K.-F., Sarigiannidis, P., e Fragulis, G. F. (2021). Hitboxes: A survey about collision detection in video games. In Fang, X., editor, *HCI in Games: Experience Design and Game Mechanics*, pages 314–326, Cham. Springer International Publishing.
- Linietsky, J., Manzur, A., e Godot Engine Community (2014). Godot engine documentation: BitMap class. https://docs.godotengine.org/en/stable/classes/class_bitmap.html#class-bitmap-method-opaque-to-polygons. Accessed on 21st March 2026.
- Mandachescu, C. A. (2003). Path finding in 2D games. Master's thesis, Concordia University, Montreal, Canada.
- Mileff, P. (2023). Collision detection in 2D games. *Production Systems and Information Engineering*, 11(3):12–31.
- Reddy, P. R., Amarnadh, V., e Bhaskar, M. (2012). Evaluation of stopping criterion in contour tracing algorithms. *International Journal of Computer Science and Information Technologies*, 3(3):3888–3894.
- Santiago, J. S. (2022). Volt engine. <https://github.com/JudsonSS/Volt-Engine>. GitHub repository.
- Thompson, K. (2023). Generate pixel perfect colliders from sprites in Godot. <https://kevinthompson.info/blog/generate-pixel-perfect-colliders-from-sprites-in-godot/>. Accessed on 21st March 2026.