Rig My Ride: Automatic Rigging of Physics-based Vehicles for Games

MELISSA KATZ, McGill University, Canada
PAUL G. KRY, McGill University, Canada
SHELDON ANDREWS, École de Technologie Supérieure, Canada and Roblox, United States

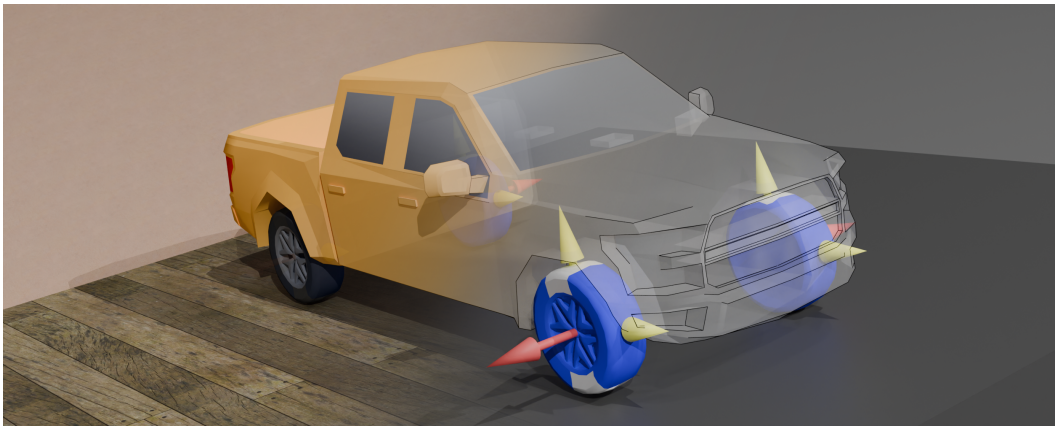


Fig. 1. Given an input mesh of a vehicle, our method automatically creates game-ready physics-based vehicle rigs, such as the PICKUP pictured here. The axes in the image represent the local hinge of each wheel. We add white strips to the wheels to make their rotation easier to see in the supplemental video.

We extend the concept of traditional rigging, which links polygonal meshes to an underlying skeleton for 3D characters, to the creation of physics-based wheeled vehicle models directly from surface geometry. Unlike character rigging, physics-based rigging involves assigning joints and collision proxies to animate the surface geometry. We present an automated pipeline that transforms a polygon soup into a physics-based, multi-wheeled vehicle model. The pipeline begins by using text-driven 2D image segmentation to identify vehicle components, which are then mapped onto the 3D mesh. A rough estimate of collision geometries and joint parameters is then used to initialize a rigid body simulation of the vehicle. Then, a numerical optimization refines these parameters in order to produce more realistic vehicle behaviour. The final result is a functioning physics-based vehicle for real-time simulations, which is demonstrated across a variety of vehicles, including cars, tricycles, lunar rovers, and even a semi-truck with 10 wheels.

CCS Concepts: • **Computing methodologies** → **Physical simulation**; *Mesh models*.

Additional Key Words and Phrases: Rigging, Physics-Based animation

Authors' Contact Information: Melissa Katz, melissa.katz@mail.mcgill.ca, McGill University, Montreal, Quebec, Canada; Paul G. Kry, kry@cs.mcgill.ca, McGill University, Montreal, Canada; Sheldon Andrews, sheldon.andrews@etsmtl.ca, École de Technologie Supérieure, Montreal, Canada and Roblox, San Mateo, United States.

ACM Reference Format:
Melissa Katz, Paul G. Kry, and Sheldon Andrews. 2025. Rig My Ride: Automatic Rigging of Physics-based Vehicles for Games. *Proc. ACM Comput. Graph. Interact. Tech.* 8, 4, Article 48 (August 2025), 23 pages. https://doi.org/10.1145/3747861

1 Introduction

The process of rigging associates the geometry of a polygonal mesh or implicit surface with an underlying articulation. While typically carried out on a 3D character, such as a humanoid or animal, where there is a clear underlying skeletal structure controlling the coarse movement and deformation of the fine-scale surface geometry, e.g., by linear blend skinning, animation of the surface geometry may also be driven by a physics-based model, such as rigid body simulation. In this context, physics-based rigging requires assigning a set of joints and bodies to animate the surface geometry.

We are specifically motivated by the need for solutions to create physics-based vehicle models, such as cars, when starting only from a 3D polygon mesh of a vehicle. The traditional process for physics-based rigging is to first segment the mesh for each component of the vehicle, e.g., the chassis and wheels. Then, a collision geometry is generated for each component that approximates the shape of the surface mesh, before mass and inertia properties are assigned. Joints are also used to attach each component to the main vehicle body (chassis). A process of iteratively resizing the collision geometry and tuning joint position and orientation is repeated until the behaviour of the vehicle is deemed to be satisfactory. The overall process is tedious and furthermore requires domain expertise. An automatic pipeline would enable both novice and experienced users to create physics-based vehicle models.

We address the issues of extensive manual tuning and requiring expert knowledge by proposing *Rig My Ride*, a two-stage process for automatically rigging physics-based vehicles. In the first stage, a text-driven 2D image segmentation is performed on the 3D surface geometry of the vehicle mesh rendered from multiple viewpoints. This allows individual components to be identified across multiple viewpoint renderings of the vehicle, which are used to segment the 3D mesh. A set of collision proxies is then estimated based on the geometry of the original mesh and the segmentation from the previous step. Furthermore, joints that couple the motion of wheels and the chassis are also estimated. Rather than iterating tediously to tune collision geometry and joint placement, the second stage of our pipeline uses a numerical optimization to refine the physics parameters of the vehicle. This is done by crafting objective functions that measure the quality of vehicle behaviour, which is evaluated using simulation roll-outs. The final result is obtained within minutes, giving a physics-based vehicle model suitable for real-time applications.

Our contribution can be summarized as a pipeline for automatically generating physics-based vehicle models that requires only an input polygonal mesh. Our approach only assumes that the up direction is known, and that the user-provided polygon surface mesh has wheels that can be reasonably approximated by a cylinder. No other properties, such as topological and geometric validity, are guaranteed in the 3D models. Figure 2 shows an overview of our system and a preview of our results.

2 Related Work

In this section, we cover related work on automatic rigging and skinning methods developed by the graphics community. Additionally, since our pipeline can be viewed as solving a computational design problem, we cover work on interactive and optimization driven algorithms for design of articulated linkages and mechanical assemblies. However, we begin by briefly presenting some of

the recent work on automatic segmentation of 3D objects using computer vision and foundational AI models, since the first phase of our pipeline segments the surface mesh of a 3D vehicle.

2.1 Part-based Segmentation

Part-based segmentation of 3D geometry has long been of interest for computer graphics researchers, and a number of approaches have been proposed based on multi-objective optimization [Simari et al. 2009], convex shape analysis [Kaick et al. 2015], and data-driven methods [Xu et al. 2016]. The survey by Rodrigues et al. [Rodrigues et al. 2018] provides an excellent summary of earlier approaches.

Recently, deep learning based approaches have proven to be successful in their ability to segment and label regions of a 3D mesh. There are two general approaches here: i) segmentation directly using features of the 3D model, ii) rendering the 3D model from multiple viewpoints and using a robust 2D image-based segmentation technique to segment regions of each rendered image. With the former class of approaches, it is common to use a point cloud generated from the surface geometry as the input. Wang et al. [Wang et al. 2020] proposed a few-shot approach that adapts a segmentation retrieved from template objects contained in a labeled point cloud dataset. Sharp et al. [Sharp et al. 2022] proposed to use network layers based on a simple diffusion operator and showed that they are effective for learning tasks involving surface geometry. Their DiffusionNet architecture is somewhat agnostic to the geometric representation, accepting polygonal elements and points as input. However, success in segmentation is demonstrated only for a limited class of objects with 3D meshes that are well-suited for diffusion. Our pipeline does not make assumptions about the quality of the provided mesh, assuming only that we have a polygon soup representing a vehicle. Other researchers have found that training networks for performing splitting, merging, and fixing operations helps generalize to out-of-distribution examples [Jones et al. 2022].

Related to the problem of segmentation of 3D models is determining the articulation between individual parts. Self-supervised [Liu et al. 2023a] and unsupervised [Xu et al. 2022a] methods have shown promise on this task, although they require an existing segmentation. Temporal sequences of part motions may also be required as input [Xu et al. 2022a; Yan et al. 2019; Yi et al. 2018]. Other approaches find kinematic hierarchies [Abdul-Rashid et al. 2021]. Our pipeline does not assume that either a part-based segmentation or an articulation are provided. We instead optimize for a vehicle model with revolute joints and an arbitrary number of wheels.

Accurate data-driven segmentation requires large, high-quality datasets of surface geometry and labeled components, and some work has focused on developing framework to facilitate the tedious labeling process [Yi et al. 2016]. While databases of high-quality segmented 3D models exist, they are often limited to a small number of exemplars per class of object and thus do not perform well on geometry for out-of-distribution examples. However, databases of segmented and annotated 2D images are much more common, and often contain orders of magnitude more training samples compared to their 3D counterparts. This has lead to the development of powerful foundational models for performing automatic segmentation of 2D images in the wild [Kirillov et al. 2023; Li et al. 2022; Lüddecke and Ecker 2022]. A growing number of methods exploit the capabilities of text-driven image foundation models to automatically perform part-level segmentation of 3D objects. The general approach here is to extract segmentation labels of multi-view rendered images of the object using pre-trained image-language models. Segmentation labels are then mapped back to a point cloud [Liu et al. 2023c; Zhou et al. 2024, 2023] or polygonal mesh [Abdelreheem et al. 2023] representation. Many authors note that it is not trivial to map 2D segmentation to 3D elements, and work continues to address the problem of resolving multi-view label consistency [Thai et al. 2024]. Our segmentation algorithm uses an approach similar to that of PartSLIP [Liu et al. 2023c], where vehicle models are rendered from multiple viewpoints, enabling GLIP [Li et al. 2022] to

identify vehicle components in each rendered view. The Segment Anything Model (SAM) [Kirillov et al. 2023] is then used to perform pixel level segmentation in these regions. More details can be found in Section 3.1. The use of these two models for rigging can be seen in Articulate AnyMesh [Qiu et al. 2025], which was developed concurrently to our own work and leverages the power of 2D segmentation techniques to create a variety of joints on a wide array of objects. However, they apply a generalized diffusion step to improve their segments, while we use a physics-based optimizer.

2.2 Automatic Rigging

The process of rigging typically involves associating the geometry of a polygonal mesh with degrees of freedom of an underlying articulated structure, i.e., a skeleton. Assigning which regions of geometry are associated with the different parts of the articulated structure can be a tedious and time consuming process. Numerous methods have been developed to reduce the time required for rigging. The Pinnochio system [Baran and Popović 2007] computes rigging weights and a base skeleton using a medial surface and provided skeleton. RigNet [Xu et al. 2020] automatically computes the underlying skeleton in addition to estimating weights. A medial axis type of representation is computed and used as input to their method, but offers little control of the specific template used for rigging. MoRig [Xu et al. 2022b] allows the user to specify the target skeleton, although requires exemplar motion sequences in the form of point clouds. Other work has similarly inferred rig parameters from mesh animation sequences [Le and Deng 2014], however this requirement does not apply to our target use case.

2.3 Computational Design of Mechanical Assemblies

Developing computational methods for designing articulated linkages and mechanical assemblies has been of interest in the graphics community for a number of years. Many previous approaches have focused on enabling non-expert users to create complex animated linkages [Coros et al. 2013; Thomaszewski et al. 2014], compliant mechanisms [Megaro et al. 2017b; Zhang et al. 2021], robotic and animated characters [Geilinger et al. 2018; Maloisel et al. 2023; Megaro et al. 2015a], cable-driven linkages [Li et al. 2017; Megaro et al. 2017a] and fabricable machines [Bächer et al. 2014; Megaro et al. 2015b]. Some researchers have noted that the parameter spaces of mechanical assemblies can be highly non-linear, thus requiring robust numerical methods. This is especially true in our application, since ground-wheel contact and optimization of the collision geometry gives a discontinuous and noisy error landscape. We therefore use a stochastic gradient-free optimization method [Hansen and Ostermeier 1996] to refine the rig parameters of our vehicle models.

3 Methodology

For the following sections, let us define our model M with axis-aligned bounding box defined by $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$, $z_{\min}$, $z_{\max}$ with center (x_c, y_c, z_c), and assume that the positive z direction defines the up direction. Our final major assumption is that the default position of the wheels would allow it to roll forward if the wheels were correctly segmented. However, we do not know in advance the correct forward direction for the vehicle.

Our system's pipeline, which will take as input a vehicle mesh and output a rig for its wheels, can be seen in Figure 2. We assume little about the initial geometry, so we perform the following pre-processing steps to bring input geometry into a consistent format. We start by triangulating the model. We then scale it such that the largest dimension becomes unit length, which helps us place the cameras to render the geometry. We finally create a 3D point cloud $\mathcal{P}$ with a fixed surface density by uniformly sampling the surface of the model. Since we control the density of the points, we can approximate the mean distance between a point and its next closest neighbour

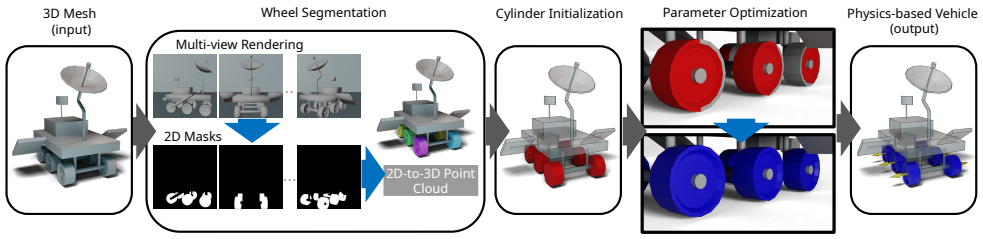


Fig. 2. Given an input mesh of a vehicle (left), here showing the ROVER model, our method automatically produces a rigged version of the vehicle mesh (right) that we can *drive* in an interactive physics-based simulation. The pipeline of our approach first uses multi-view rendering with a prompt based segmentation to produce 2D wheel masks via GLIP and SAM. Masks are projected onto the model and clustered to identify sets of 3D points that make up the wheels. Cylinders are fit to the point sets, and a gradient free minimization uses physics simulation to optimize the wheel geometry and joint constraints so as to produce a functional rig for interactive simulation.

in the point cloud, $\bar{d}$, as the square root of one over the density. In our case, we settled on a very dense point cloud of 300,000 points per unit squared, which results in approximately 0.002 units of length between points.

3.1 Wheel Segmentation

Finding the wheels on the model is our first challenge. Our approach is to render the model from different viewpoints and identify which pixels in those images are the wheels. As such, we aim the camera at the center of the mesh bounding box, and position it at the same height as the center of the bounding box, at a distance such that a unit cube at the center of the bounding box perfectly fits into its horizontal field of view. We then rotate the camera about the z-axis of the model to create rendered RGB images.

For each image, we use a two step process to identify which pixels make up the wheels. We first use GLIP [Li et al. 2022], which allows us to input our images and the prompt "wheel of a vehicle" to obtain a set of bounding boxes for the wheels that appear in each image. We then pass each image along with corresponding bounding boxes to SAM (segment anything) [Kirillov et al. 2023] to obtain a set of masks which we merge together for each image. An example of such a mask can be seen in Figure 3.

Identifying what parts of the 3D input geometry correspond to wheels becomes straightforward given the masks produced by SAM. Since we know the position from which each image was rendered, we cast a ray in the direction of each pixel in the mask and find the intersection with the model geometry to compute the visible point in $\mathbb{R}^3$ on the surface of the model. We then find the point in $\mathcal{P}$ which is closest to this visible point and flag it as being part of a wheel. Once this process has been applied from every view, we then cluster all of the flagged points by using DBSCAN [Ester et al. 1996], where we set the radius defining the neighbourhood of a point as $\bar{d}$ defined by the density of our point cloud. This gives us large clusters for the wheels, but can also create some extra *noise* clusters which typically come from imperfections in the mask. We eliminate these extra clusters by ignoring those that are too far away from $z_{\min}$. We observe that a threshold of $5\bar{d}$ works well across all of our vehicle models.

From this process, multiple crucial hyperparameters emerge: the number of images of the vehicle that must be rendered, the need to include different viewpoints (such as the underside of the vehicle), the resolution of each image, the prompt that we give to GLIP, and finally the confidence

Fig. 3. Sample output of both GLIP (left) and SAM (right), overlayed on an example input image.

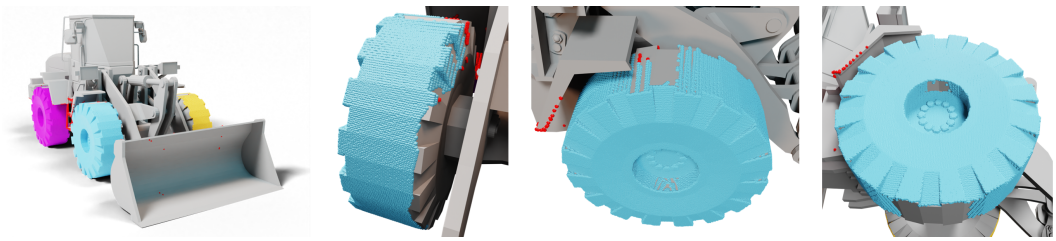


Fig. 4. (Left) The BULLDOZER model in an example view, with wheels highlighted, and examples of the wheel parts that are often never seen in the renders we use to find geometry that makes up the wheels. From left to right: the inner part of the wheel near the chassis, the top of the wheel, and the bottom of the wheel.

that GLIP must reach in order to validate its bounding boxes as having fulfilled our prompt. We have explored a variety of combinations of these hyperparameters, and our results can be viewed in appendix A. In short, we have elected to use 10 square images 700 pixels in width, with a simple revolution around the vehicle, and prompting GLIP with the phrase "wheel of a vehicle" while requiring it to achieve 50% confidence in its guess. With these parameters, and with DBSCAN's 3D segmentation as described above, we are able to create N point clusters that accurately reflect the real number of wheels of each vehicle. However, these clusters only capture about 70% of the actual area of each wheel, because some parts were not visible on any rendered image, especially any parts that were obstructed by the chassis. We demonstrate some common missed areas in Figure 4. Furthermore, these segments do not give us any information about the kinematic chain created between the chassis and the wheels. Thus, we need to optimize to capture the entire wheel shape and to find the way that the wheels connect to the chassis.

3.2 Cylinder Initialization

We choose to represent the changes to the wheel geometry by the vertices that get included or excluded from each wheel. However, we need to transform this complex integer programming task into a form more amenable to optimization. To this end, we use cylinders to approximate the wheels: we then run an intersection test between each point and the cylinder in order to determine

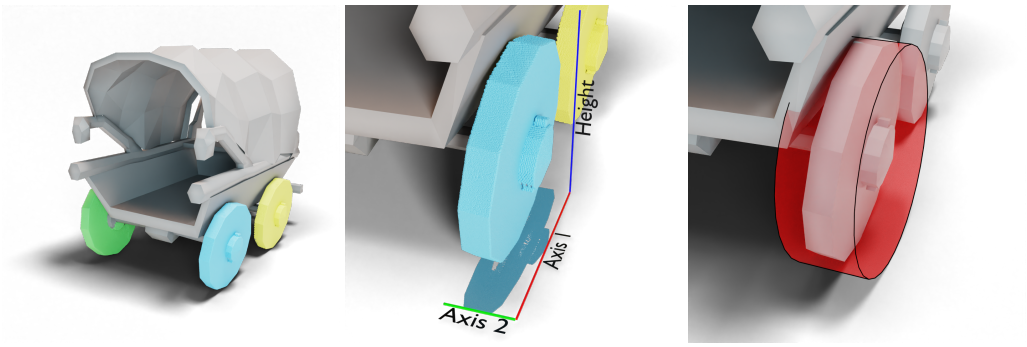


Fig. 5. (Left) A CARRIAGE model with slanted wheels poses an interesting challenge for vehicle rigging. (Center) Principal component axes of the planar projection of the wheel: here we choose axis 2 as the direction of the initial cylinder axis because axis 1 more closely matches the wheel height. (Right) The result of the cylinder initialization process, ready for optimization.

which vertices belong to that wheel. Thus, our optimization task becomes finding the best axis, position, radius, and length of each cylinder.

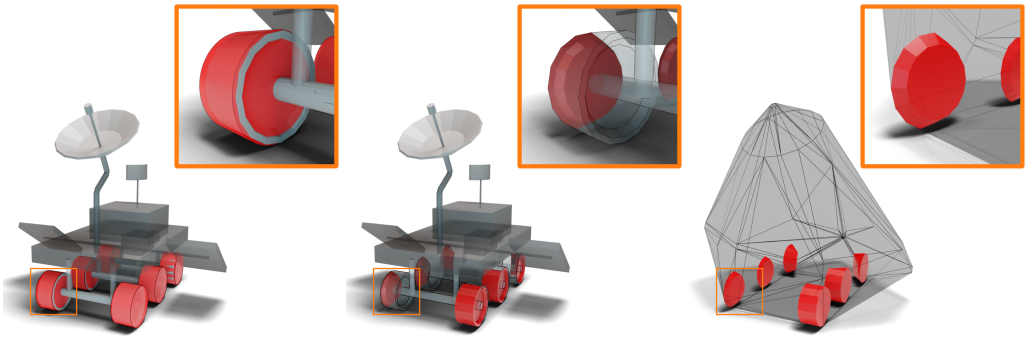
Before we can start our optimization, we must thus transform our point clouds into initial cylinder guesses. Many cylinder fitting algorithms exist, but few address the problem of finite (capped) cylinders, and instead assume that most input samples belong to the curved surface of a cylinder. In contrast, our point clouds contain a lot of samples from what would be seen as cylinder caps (e.g., hub caps of a wheel). However, since we know that the axis of the wheels will be perpendicular, or close to perpendicular, to the up-direction (z), we can initialize them using a heuristic approach.

As a first step, we eliminate the z-component of each point in the point cloud, in order to make the data 2D. This will eliminate most of the noise created by the lack of data from the renders and create a clearer projection of the curved surface. Next, we perform principal component analysis on these points, which gives us a bounding box for the wheel which is axis-aligned in the z (up) direction and oriented in the x and y directions. This gives us all the cylinder parameters that we need. The diameter is given by the oriented side of the bounding box which is most similar to the axis-aligned side, the length is determined by the remaining oriented side, the position is simply the center of the bounding box, and the axis is represented by the principal component responsible for the length of the cylinder. An example of a possible bounding box and the resulting cylinder can be seen in Figure 5.

Once we obtain all the cylinders, we eliminate any cylinder that is of a significantly smaller order of magnitude than the other ones, which removes any outliers that were not eliminated by the clustering step.

3.3 Parameter Optimization

We want to evaluate our segmentation based on the performance of the model in a rigid body simulation. To this end, using the base model as reference, we create $N + 1$ bodies, one for each wheel and one for all the remaining points, which will make up the chassis. Next, we create N revolute joint constraints using the central axis of each cylinder. However, we will not be using the cylinders as collision geometry, since they will poorly estimate the visual quality of each segment, and we will instead use the convex hulls of the $N + 1$ vertex sets. These convex hulls will also determine the position of each joint. The issue with cylinders as collision proxies, and the solution



(a) Initial cylinders (b) Geometry inside cylinder (c) Convex hull inside cylinder

Fig. 6. An illustration of the issue with using cylinders as a collision proxy. (a) Height-wise, the initial cylinders appear well aligned with the correct axis of rotation. These collision proxies will lead to near-perfect rolling, giving accurate physical behaviour. However, we observe that the wheel geometry is not fully enveloped by the cylinder due to small perturbations of the cylinder's center, as shown in the highlighted example. (b) Further examination of the geometry enveloped by the cylinder reveals that the cylinder fails to capture all the vertices we expect to be part of the wheel. As such, the good rolling behaviour is misaligned with each wheel's poor visual appearance. (c) The convex hull of this geometry will directly cause poor rolling behaviour, fixing the misalignment of the cylinder proxies.

(a) A wheel with a high vertex count, and optimized cylinder (b) Geometry inside cylinder. (c) Geometry inside cylinder, after dilation-erosion steps.

Fig. 7. An illustration of how convex hulls can patch over holes in geometry. In the case of this cylinder, all the vertices near the floor were correctly captured, but some vertices on the top and side, that have little influence on movement due to the wheel's high vertex count, were excluded. Including some of the neighbouring vertices was able to mitigate the issue.

offered by convex hulls, are illustrated in Figure 6. To counteract the issue of convex hulls patching over small holes in geometry, we slightly dilate each wheel by including the neighbouring vertices of the segment, immediately eroding it afterwards to limit the number of new vertices that we add. An explanation of hole-patching and the fix created by dilation-erosion can be seen in Figure 7. Finally, we will make our vehicle travel in the direction which is perpendicular to the up-direction and the average direction of the cylinders.

We found that the cylinder parameters could not be found with gradient based methods, since such a simulation would not have an obvious derivative. Furthermore, in cases were small changes in cylinder parameters lead to large changes in wheel geometry, roll-outs would frequently diverge. For this reason, we rely on CMA-ES [Hansen and Ostermeier 1996] for our optimizer, which is gradient-free. At each step, we initialize the rigid bodies as described above. We also set the vehicle's desired motion direction $\hat{\mathbf{d}}$ by summing the axes of all the cylinders (axis flipped if it points in the opposite direction to the running total) and calculating the cross product with the up (z) vector. Finally, we give each wheel an initial velocity in $\hat{\mathbf{d}}$'s direction and let the simulation roll out as CMA-ES evaluates a fitness function.

Gradient-free minimization will randomly sample many possible solutions. However, before running a simulation to compute a score, we reject a given set of parameters based on three heuristics that identify problematic situations. Specifically, we immediately reject when there is a cylinder that is (1) initially too far from the ground, (2) not contained in the vehicle bounding box, or (3) intersecting another cylinder. Rejection is handled by returning a very large penalty.

With the remaining samples, we run a simulation for Q frames and compute a fitness function consisting of four parts: travel (f_t), compliance (f_c), boundary quality (f_b), and mean squared error (MSE), (f_e). We assign weights to these terms, experimentally determined in advance, to ensure that they properly penalize poor solutions, and likewise, to ensure that they are of similar magnitude to one another.

Travel. We compute the total distance traveled between the start frame 1 and the end frame Q using the position $\mathbf{p}$ of the chassis projected onto the desired motion vector $\hat{\mathbf{d}}$. Since we must minimize our objective function, we convert this distance into a penalty f_t by inverting its absolute value. If the vehicle ends up moving backwards during the simulation, the candidate is rejected; otherwise, we assign

$$f_\mathrm{t} = \frac{1}{\left|(\mathbf{p}_Q - \mathbf{p}_1) \cdot \hat{\mathbf{d}}\right|}. \tag{1}$$

Compliance. We compute a penalty for what we call compliance, f_c, which will be a measure of similarity between the cylinder and the convex hull of wheel j, examined through their respective volumes V_{C_j} and V_{H_j}. The best case would consist of the cylinder and the convex hull having an identical volume, thus we look at how far the ratio lies from 1. Since we grow the geometry inside each segment, there may be cases where the V_{H_j} exceeds V_{C_j}, in which case we flip the ratio,

$$f_\mathrm{c} = \frac{1}{N}\sum_{j=1}^{N} 1 - \frac{\min(V_{H_j}, V_{C_j})}{\max(V_{H_j}, V_{C_j})}. \tag{2}$$

Boundary. Even though one wheel may be made up of several disconnected components, one connected component does not generally belong to both the wheel and the chassis. As such, if we do split a connected component, these splits must be kept to a minimum. We define a vertex as *fringe* if it has at least one neighbour that's part of the chassis and at least one that's part of the wheel. We then count the number of loops that these fringe vertices create on the model:

$$f_\mathrm{b} = \frac{1}{N}\sum_{j=1}^{N} (\text{Loops on wheel } j). \tag{3}$$

MSE. Because the original cylinders we compute in Section 3.2 tend to be a high quality representation of the wheels, we compute a mean squared error term to encourage each optimized

Fig. 8. Left shows bounding boxes found by GLIP for parts on the "front of the vehicle", and right shows points on the vehicle (green) inside these boxes that we combine to compute the forward direction (blue). This permits the straightforward addition of a steering rig to the front wheels of the vehicle.

cylinder to have a similar length l, radius r, and center position p as the corresponding parameters $\bar{l}$, $\bar{r}$, $\bar{p}$, in the original cylinder. This term is a simple sum of squares for each wheel,

$$f_e = \frac{1}{N}\sum_{j=1}^{N}(r_j - \bar{r}_j)^2 + \|p_j - \bar{p}_j\|^2 + (l_j - \bar{l}_j)^2. \tag{4}$$

Once CMA-ES runs for a fixed number of generations, we return each wheel's updated list of vertices, along with its final joint position.

3.4 Extension: Steering

The above sections describe a method for automatically creating a vehicle that moves forward. However, this technique can be easily extended to include more complex motion, by adding steering for example. To this end, we prompt GLIP for the "front of the vehicle" on each image. We do not give these bounding boxes to SAM, and we instead directly send a ray through each of their centers. We then find the median hit location which we use to define a front direction vector $\hat{f}$ between it and the center of the vehicle bounding box. An example of this procedure can be found in Figure 8. We can then flip the desired direction of motion $\hat{\mathbf{d}}$ depending on the sign of their dot product. Using the correctly flipped vector $\hat{\mathbf{d}}$, we can identify the front wheels as those furthest along that direction, and add additional revolute joint constraints to these wheels to equip the vehicle with a simple steering setup.

3.5 Extension: Suspension

While steering was made possible by creating new prompts, other extensions are possible by running a supplementary optimization steps. We demonstrate this by adding a suspension system to our vehicle. After initializing our rigid body model with the optimized chassis and wheels, we create prismatic joints and springs with stiffness k, damping b, and rest length l_0 between the chassis and the wheels. The vehicle is then made to drive through rough terrain for Q frames with a constant velocity while computing a fitness function with two parts: stability (f_s) and contact (f_n).

Stability. Suspension exists to make the chassis stable, even on bumpy terrain, which translates to minimizing the rotation of the chassis. We evaluate this by measuring two dot products, one between the up direction at initialization $\hat{\mathbf{y}}$ and the current frame $\mathbf{y}_q$, the other between front direction at initialization $\hat{\mathbf{d}}$ and the current frame $\mathbf{d}_q$, in order to obtain the penalty

$$f_s = \sum_{q=1}^{Q} 1 - \mathbf{y}_q \cdot \hat{\mathbf{y}} + 1 - \mathbf{d}_q \cdot \hat{\mathbf{d}}. \tag{5}$$

Contact. If suspension becomes too weak, the chassis can start to graze against the ground. We counteract this tendency by summing up the largest penetration distance Δp_q between the chassis and the ground at each frame, which yields us

$$f_n = \sum_{q=1}^{Q} \Delta p_q. \tag{6}$$

4 Results

We demonstrate the results of our system on a variety of vehicle models found on TurboSquid, CGTrader, and Thingiverse (see Figure 13). When selecting the 3D models, it was important to us to find a variety of wheel counts, topologies, and styles. In the end, our models contain between 2 and 10 wheels, can contain wheels defined by as little as 96 vertices or by as many as tens of thousands.

4.1 Implementation Details

Multi-view renderings of each 3D model are generated in Blender using a custom Python script. For simulations, we use a custom rigid body simulator written in C++, which allows us to gain efficiency and parallelization while performing simulation rollouts during optimization. Full source code for our pipeline is available at https://github.com/Melissa2661/RigMyRide.

We use GLIP-L to find the bounding boxes for our experiments and SAM ViT-H to convert the boxes into masks. We use the pre-trained weights offered in their respective papers, with no additional training on our end. The two prompts that we used were "wheel of a vehicle" for wheel initialization and "front of a vehicle" for steering; we attempted to prompt GLIP for the "back of a vehicle", but this yielded almost no bounding boxes.

For CMA-ES, we optimize for 300 generations, where each candidate runs a simulation for 300 steps to evaluate the fitness function. Each generation yielded $32 \times N$ candidates; we chose to take into account the number of wheels in order to counteract the increased difficulty in optimizing the increased number of parameters. The initialization of CMA's covariance matrix depends on the parameter: each cylinder's position yields a 0.01 in the matrix, while direction, depth and radius creates a 0.03.

4.2 Pipeline Output

Examples showing the rigs before and after optimization are shown in Figure 13. Likewise, please see the supplementary video for simulations of the vehicles shown in the figure. A sample output of each stage of our pipeline can be seen in Figure 2.

We report the time each major section of our pipeline takes in Table 1. Note that during 2D segmentation, GLIP and SAM took on average 1.9 and 1.1 seconds, respectively, to analyze each image. Unsurprisingly, optimization is the most time-consuming part of our pipeline; however, if the vehicles have a sufficiently low face and vertex count, the process can take less than 10 minutes total. As such, it could be interesting to restart the optimization process multiple times and pick the

Table 1. Timing of each major section of our pipeline: the creation of the 3D point cloud, the casting of rays from the 2D mask onto the 3D model, initialization of the cylinders, and optimization of cylinders with CMA-ES. Unless otherwise indicated, all times are measured in seconds.

Name	Face count	PC creation	2D - 3D transfer	Cylinder init	Optimization	
Bicycle	126 889	0.30	31	0.1	322	(5.4 min)
Bulldozer	18 414	0.05	16	0.8	168	(2.8 min)
Bus	25 666	0.05	23	0.2	217	(3.6 min)
Carriage	1 552	0.03	22	2.5	62	(1.0 min)
Cement mixer	8 698	0.03	24	1.0	237	(4.0 min)
Old pickup	4 824	0.03	24	1.9	144	(2.4 min)
Racecar	31 798	0.05	26	3.0	579	(9.7 min)
Rover	2 828	0.03	22	0.3	160	(2.7 min)
Scooter	41 265	0.13	25	0.1	132	(2.2 min)
Suspension	188 450	0.38	32	1.2	5078	(84 min)
Taxi	4 185	0.05	20	0.9	98	(1.6 min)
Tricycle	26 724	0.04	26	1.4	133	(2.2 min)
Truck	39 792	0.15	25	0.5	1226	(20.4 min)

best result from them, which could allow some vehicles to be better segmented. In contrast, little to no variation in results was found in the rest of the system, so these results may be easily cached.

We report qualitative results of our segmentation in Table 2, showing the difference in segment accuracy between the 3D segmentation step and the final output post-optimization. We created ground-truth segmentations for 13 vehicles, and compare the total area captured by each segment at both steps.

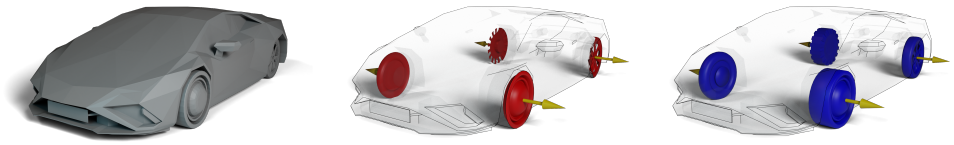
For the segmentation step, we can see that every wheel gets its own segment, with the exception of very close co-axial wheels as seen in the CEMENT MIXER vehicle. However, having these wheels fuse together has minimal impact on the accuracy of that vehicle's motion; a much more detrimental fusion would have been between two physically close wheels that do not share an axis, like the ones on the TRUCK model. Such a situation was avoided thanks to the high accuracy of the masks returned by SAM and our dense point cloud, which recognized each wheel as a distinct entity. Even though the number of segments is generally optimal, the information captured by each segment generally is not, with the vehicles averaging 70.2% accuracy for the wheels.

However, once we allow the wheels segments to be optimized, the average accuracy increases to 91.1%. Furthermore, in 9 out of 13 vehicles, 100% of the wheel area was captured, and 5 out of 9 capture no area from the chassis, resulting in perfect segments. Only two vehicles suffer from the optimization process, namely the BICYCLE and the SCOOTER. Since they both only have two thin wheels, they have trouble balancing during the physics-based simulation; as such, the optimizer has trouble converging.

Finally, there are three vehicles that show improved, but not optimal, segments, the CEMENT MIXER, TAXI and TRICYCLE. On the last two models, the wheels are very close to their wheel wells. As such, the surrounding geometry gets easily pulled into the wheel segments; adding texture to these models in order to improve the initial segmentation does not mitigate this effect. However, since they are still able to roll, the optimizer is able to improve on the initial segments. Meanwhile, for the CEMENT MIXER, the lack of optimality occurs due to two wheels being completely excluded

Table 2. Comparison of our method after initial segmentation and after final optimization against ground truth. In our context, a false positive refers to a part of the chassis that got incorrectly labeled as a wheel.

Model	Wheel count		Segmentation		Optimization	
	Ground truth	Created	True Positive	False Positive	TP	FP
Bicycle	2	2	77.6%	10.6%	65.7%	13.9%
Bulldozer	4	4	71.7%	0.0%	100 %	0.0%
Bus	4	4	72.4%	0.6%	100 %	0.0%
Carriage	4	4	83.3%	0.2%	100 %	0.0%
Cement mixer	10	6	74.5%	1.5%	80.2%	0.1%
Old pickup	4	4	94.0%	5.3%	100 %	1.8%
Racecar	4	4	60.7%	1.3%	100 %	3.0%
Rover	6	6	94.0%	5.2%	100 %	0.0%
Scooter	2	2	71.7%	0.2%	50.6%	1.7%
Suspension	4	4	60.0%	3.5%	100 %	2.5%
Taxi	4	4	60.6%	4.3%	100 %	2.0%
Tricycle	3	3	78.3%	14.8%	87.7%	6.5%
Truck	10	10	68.1%	0.9%	100 %	0.0%
		Average	70.2%	3.3%	91.1%	2.4%



(a) Global vehicle view (b) Initial wheel segments (c) Final wheel segments

Fig. 9. Output for the FRANKENWAGON vehicle.

from the segments; as they were disconnected components, the optimizer was able to find a good enough solution without including them.

4.3 Special Case: Asymmetric Wheels

As a stress test, we demonstrate how our pipeline handles a vehicle with highly asymmetric wheels, the FRANKENWAGON model. Such a vehicle would not be possible to rig with a closed-form equation for the wheels. The result can be seen in Figure 9.

4.4 Steering

Our technique for identifying the front of the vehicle works well when the vehicle's shape creates no ambiguity. However, some vehicle topology does make it hard to differentiate between front and back, which leads to GLIP identifying the back of the vehicle as its front. Since this step only changes the sign of the desired direction of motion, such errors can be easily undone by the user in one key press.

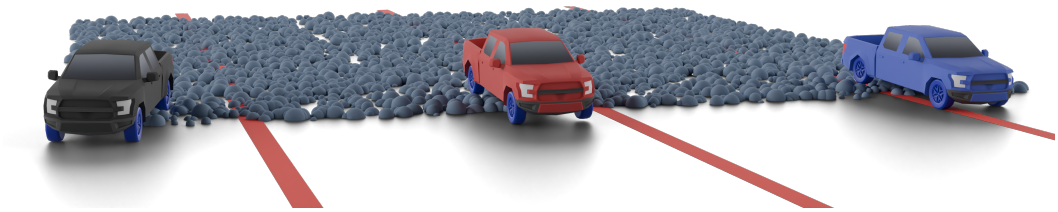


Fig. 10. Final pose of some pickups after driving through rough terrain for 430 frames. From left to right: vehicle with no suspension, one with suspension initialized to maximal stiffness/damping; one with optimized suspension. The red strips on the ground align with the chassis' position on the first frame.

4.5 Suspension

In the extended example, the vehicle suspension was optimized 150 iterations, where each generation created 10 candidates and each candidate ran the simulation for 300 steps. Furthermore, to reduce the large difference in magnitude between the rest length and the stiffness and damping parameters, we instead optimize for $k_\mathbf{e}$ and $b_\mathbf{e}$, such that the spring's stiffness and damping are $10^{k_\mathbf{e}}$ and $10^{b_\mathbf{e}}$ respectively. The coefficients $k_\mathbf{e}$ and $b_\mathbf{e}$ are bounded to the range [0.0, 9.0], while the rest length is bounded to between 0 and the diameter of the wheel. The performance of the resulting suspension is demonstrated in Figure 10.

4.6 Ablation of Fitness Function

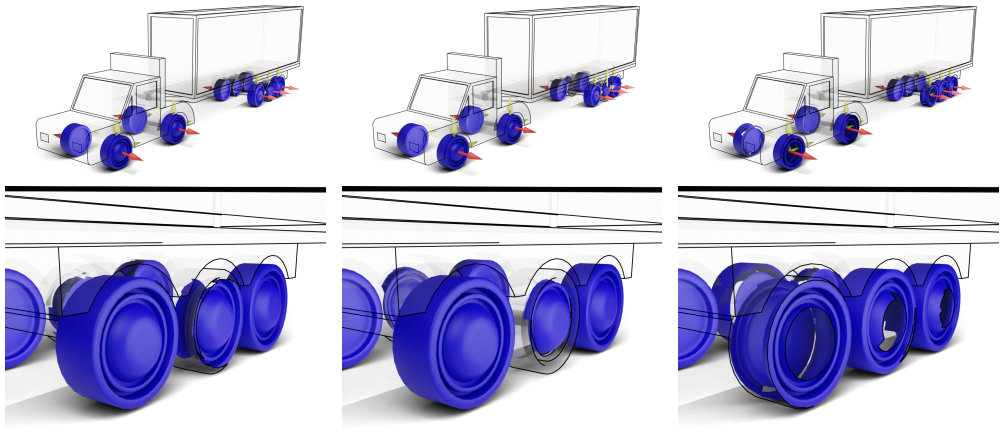
We conduct an ablation study to demonstrate the necessity of each penalty. The qualitative results can be seen in Figures 11. We omit mean squared error from the ablation study as it only affects the convergence of the optimizer, with little effect on the quality of the segments. We see that distance and compliance help ensure that all the wheels contribute to forward motion, since the lack of these penalties leads to vehicles where some of the wheels may not even touch the ground, while the border penalty prevents holes in the wheel geometry.

5 Limitations and Future Work

One limitation of our work is related to the assumption that wheels can be approximated into cylinders. Even though this is very often the case, the assumption does not hold for spherical wheels and provides a sub-optimal approximation of the smooth curves of a tire. A more general alternative here may be to use superquadrics instead of cylinders for the collision geometry optimization. Even though there has been recent work with segmenting meshes into superquadrics [Liu et al. 2023b; Wu et al. 2022] and even segmentation using SAM [Liu et al. 2024], our preliminary investigations on incorporating quadric optimization into our pipeline yielded unsatisfactory segments.

Our system is also unable to recognize additional components that may require articulation due to motion of the wheels. This is most evident in the case of the TRICYCLE: its pedals should be put into motion by the rotation of the front wheel, but that is not the case. However, as can be seen in Figure 12, this presents its own topological challenges that cannot be solved with the help of GLIP or SAM, at least not in their current state.

Finally, since the optimizer evaluates physics parameters by rolling the vehicle forward, some vehicles may have trouble staying upright and thus perform poorly even if their collision geometry and joint axes are good. This is mainly a disadvantage for bicycles, but not all two-wheeled vehicles



(a) No distance (b) No compliance (c) No border

Fig. 11. Comparison between the states of the simulation after 300 steps, where selected penalties are omitted from the optimization, on the TRUCK model. With all penalties active, our optimization process returns perfect segments.

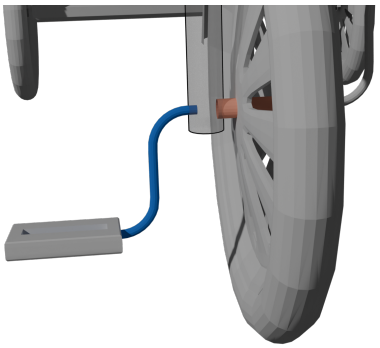


Fig. 12. The pedals of the TRICYCLE do not spin along with the wheel in our result, but they are likewise far from the wheel. Any heuristic using geodesics would erroneously pull the fork of the TRICYCLE into the segment before reaching the pedals.

suffer from this. The STEAMROLLER, for example, has no issues staying upright and is thus rigged properly (see Figure 13). We did not find a satisfactory way to mitigate this issue for bicycles.

One area that should be explored further includes finding better ways of fighting convex hull hole patching on models with high vertex counts. One fix might be to create a simplified version of the vehicle using isotropic remeshing, running the inside-outside test on cylinders using that simplified model, then transferring this information back on the full model to create the convex hulls. However, since our work focuses on vehicles used in games, which tend to be low-poly, we did not explore this idea in depth.

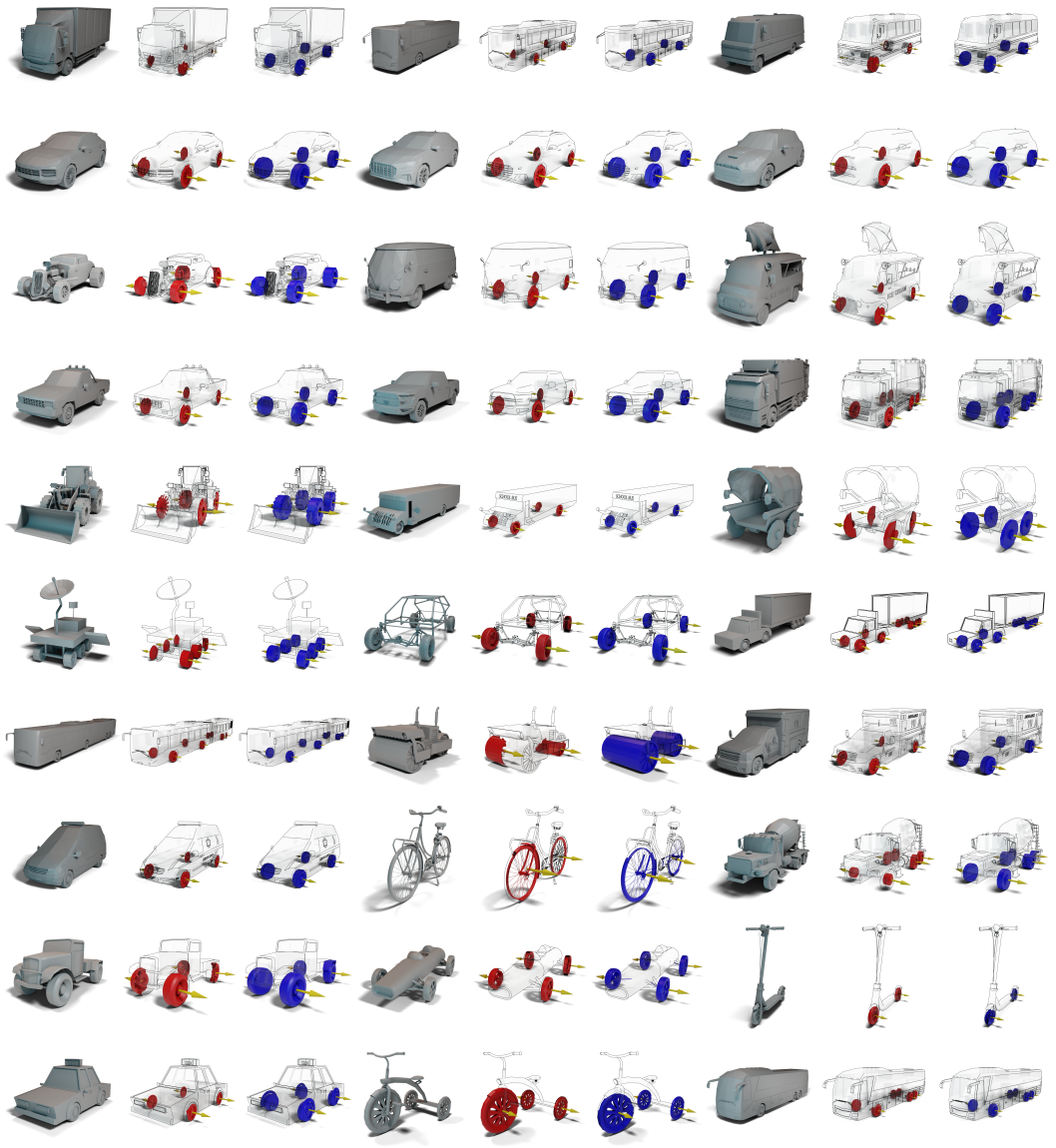


Fig. 13. Input meshes, initial wheels (red), and final wheels (blue) for a wide collection of vehicles processed by our pipeline. Notice how the initial revolute axes (yellow arrows) and wheel collision geometries are improved by the optimization stage of our pipeline. The last 8 vehicles, starting with the bicycle, are cases where the final geometry is not optimal.

6 Conclusions

The Rig My Ride system proposes a way to use the strengths of prompt-driven 2D image segmentations and applies them not only to 3D segmentation tasks, but also to physics-based animation. We also leverage numerical optimization with simulation-in-the-loop to refine rig parameters and produce physical behaviour that is consistent with the input geometry. Our system is able to create

rigs of vehicles without any human input, which not only allows the user to skip this tedious step, but also makes it possible to rig large databases of vehicles without any supervision and without knowledge about the number of wheels or the topology of the vehicle mesh. Rig My Ride can also be enhanced for even richer motion by adding a suspension system through a supplementary optimization step. Furthermore, if the vehicle presents enough characteristics to distinguish its front from its back, it can automatically determine the appropriate steering axes.

Incorporating other geometries as part of the optimization stage is also interesting future work. For instance, once superquadric segmentation becomes more advanced, our technique could be easily expanded to adapt to more general convex geometry, which creates the opportunity to articulate more complex mechanisms like crane arms. Such segmentations, paired with a soft body simulation, could also be used to rig vehicles with treads. With minor tweaks to the optimization function, our simulation-based verification technique could also be used in combination with generative techniques, e.g., Articulate AnyMesh [Qiu et al. 2025], in order to reduce interpenetrations and holes, leading to more realistic results.

Acknowledgments

The authors would like to thank Hsueh-Ti Derek Liu and Victor Zordan for providing valuable feedback and discussions.

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Ahmed Abdelreheem, Ivan Skorokhodov, Maks Ovsjanikov, and Peter Wonka. 2023. SATR: Zero-Shot Semantic Segmentation of 3D Shapes. arXiv:2304.04909 [cs.CV]

Hameed Abdul-Rashid, Miles Freeman, Ben Abbatematteo, George Konidaris, and Daniel Ritchie. 2021. Learning to Infer Kinematic Hierarchies for Novel Object Instances. arXiv:2110.07911 [cs.CV]

Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. 2014. Spin-it: optimizing moment of inertia for spinnable objects. *ACM Trans. Graph.* 33, 4, Article 96 (July 2014), 10 pages. doi:10.1145/2601097.2601157

Ilya Baran and Jovan Popović. 2007. Automatic rigging and animation of 3D characters. *ACM Trans. Graph.* 26, 3 (July 2007), 72–es. doi:10.1145/1276377.1276467

Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. 2013. Computational design of mechanical characters. *ACM Trans. Graph.* 32, 4, Article 83 (July 2013), 12 pages. doi:10.1145/2461912.2461953

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (Portland, Oregon) *(KDD'96)*. AAAI Press, 226–231. https://api.semanticscholar.org/CorpusID:355163

Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski, and Stelian Coros. 2018. Skaterbots: optimization-based design and motion synthesis for robotic creatures with legs and wheels. *ACM Trans. Graph.* 37, 4, Article 160 (July 2018), 12 pages. doi:10.1145/3197517.3201368

Nikolaus Hansen and Andreas Ostermeier. 1996. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. 312 - 317 pages. doi:10.1109/ICEC.1996.542381

R. Kenny Jones, Aalia Habib, and Daniel Ritchie. 2022. SHRED: 3D Shape Region Decomposition with Learned Local Operations. *ACM Trans. Graph.* 41, 6, Article 186 (Nov. 2022), 11 pages. doi:10.1145/3550454.3555440

Oliver Van Kaick, Noa Fish, Yanir Kleiman, Shmuel Asafi, and Daniel Cohen-OR. 2015. Shape Segmentation by Approximate Convexity Analysis. *ACM Trans. Graph.* 34, 1, Article 4 (Dec. 2015), 11 pages. doi:10.1145/2611811

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. 2023. Segment Anything. arXiv:2304.02643 [cs.CV]

Binh Huy Le and Zhigang Deng. 2014. Robust and accurate skeletal rigging from mesh sequences. *ACM Trans. Graph.* 33, 4, Article 84 (July 2014), 10 pages. doi:10.1145/2601097.2601161

Jian Li, Sheldon Andrews, Krisztian G. Birkas, and Paul G. Kry. 2017. Task-based design of cable-driven articulated mechanisms. In *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication* (Cambridge, Massachusetts) *(SCF '17)*. Association for Computing Machinery, New York, NY, USA, Article 6, 12 pages. doi:10.1145/3083157.3083161

Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, Kai-Wei Chang, and Jianfeng Gao. 2022. Grounded Language-Image Pre-training. arXiv:2112.03857 [cs.CV]

Anran Liu, Cheng Lin, Yuan Liu, Xiaoxiao Long, Zhiyang Dou, Hao-Xiang Guo, Ping Luo, and Wenping Wang. 2024. Part123: Part-aware 3D Reconstruction from a Single-view Image. arXiv:2405.16888 [cs.GR]

Jiayi Liu, Ali Mahdavi-Amiri, and Manolis Savva. 2023a. PARIS: Part-level Reconstruction and Motion Analysis for Articulated Objects. arXiv:2308.07391 [cs.CV]

Minghua Liu, Yinhao Zhu, Hong Cai, Shizhong Han, Zhan Ling, Fatih Porikli, and Hao Su. 2023c. PartSLIP: Low-Shot Part Segmentation for 3D Point Clouds via Pretrained Image-Language Models. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 21736–21746. doi:10.1109/CVPR52729.2023.02082

Weixiao Liu, Yuwei Wu, Sipu Ruan, and Gregory S. Chirikjian. 2023b. Robust and Accurate Superquadric Recovery: a Probabilistic Approach. arXiv:2111.14517 [cs.CV]

Timo Lüddecke and Alexander S. Ecker. 2022. Image Segmentation Using Text and Image Prompts. arXiv:2112.10003 [cs.CV]

Guirec Maloisel, Christian Schumacher, Espen Knoop, Ruben Grandia, and Moritz Bächer. 2023. Optimal Design of Robotic Character Kinematics. *ACM Trans. Graph.* 42, 6, Article 195 (Dec. 2023), 15 pages. doi:10.1145/3618404

Vittorio Megaro, Espen Knoop, Andrew Spielberg, David I. W. Levin, Wojciech Matusik, Markus Gross, Bernhard Thomaszewski, and Moritz Bächer. 2017a. Designing cable-driven actuation networks for kinematic chains and trees. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '17)*. Article 15, 10 pages. doi:10.1145/3099564.3099576

Vittorio Megaro, Bernhard Thomaszewski, Damien Gauge, Eitan Grinspun, Stelian Coros, and Markus Gross. 2015a. ChaCra: an interactive design system for rapid character crafting. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Copenhagen, Denmark) *(SCA '14)*. 123–130. doi:10.2312/sca.20141130

Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. 2015b. Interactive design of 3D-printable robotic creatures. *ACM Trans. Graph.* 34, 6, Article 216 (Nov. 2015), 9 pages. doi:10.1145/2816795.2818137

Vittorio Megaro, Jonas Zehnder, Moritz Bächer, Stelian Coros, Markus Gross, and Bernhard Thomaszewski. 2017b. A computational design tool for compliant mechanisms. *ACM Trans. Graph.* 36, 4, Article 82 (July 2017), 12 pages. doi:10.1145/3072959.3073636

Xiaowen Qiu, Jincheng Yang, Yian Wang, Zhehuan Chen, Yufei Wang, Tsun-Hsuan Wang, Zhou Xian, and Chuang Gan. 2025. Articulate AnyMesh: Open-vocabulary 3D Articulated Objects Modeling. *arXiv preprint arXiv:2502.02590* (2025).

Rui S. V. Rodrigues, José F. M. Morgado, and Abel J. P. Gomes. 2018. Part-Based Mesh Segmentation: A Survey. *Computer Graphics Forum* 37, 6 (2018), 235–274. doi:10.1111/cgf.13323

Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov. 2022. DiffusionNet: Discretization Agnostic Learning on Surfaces. *ACM Trans. Graph.* 41, 3, Article 27 (March 2022), 16 pages. doi:10.1145/3507905

P. Simari, D. Nowrouzezahrai, E. Kalogerakis, and K. Singh. 2009. Multi-objective shape segmentation and labeling. *Computer Graphics Forum* 28, 5 (2009), 1415–1425. doi:10.1111/j.1467-8659.2009.01518.x

Anh Thai, Weiyao Wang, Hao Tang, Stefan Stojanov, Matt Feiszli, and James M. Rehg. 2024. 3x2: 3D Object Part Segmentation by 2D Semantic Correspondences. arXiv:2407.09648 [cs.CV] https://arxiv.org/abs/2407.09648

Bernhard Thomaszewski, Stelian Coros, Damien Gauge, Vittorio Megaro, Eitan Grinspun, and Markus Gross. 2014. Computational design of linkage-based characters. *ACM Trans. Graph.* 33, 4, Article 64 (July 2014), 9 pages. doi:10.1145/2601097.2601143

Lingjing Wang, Xiang Li, and Yi Fang. 2020. Few-Shot Learning of Part-Specific Probability Space for 3D Shape Segmentation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4503–4512. doi:10.1109/CVPR42600.2020.00456

Yuwei Wu, Weixiao Liu, Sipu Ruan, and Gregory S. Chirikjian. 2022. Primitive-based Shape Abstraction via Nonparametric Bayesian Inference. arXiv:2203.14714 [cs.CV]

Kai Xu, Vladimir G. Kim, Qixing Huang, Niloy Mitra, and Evangelos Kalogerakis. 2016. Data-driven shape analysis and processing. In *SIGGRAPH ASIA 2016 Courses* (Macau) *(SA '16)*. Article 4, 38 pages. doi:10.1145/2988458.2988473

Xianghao Xu, Yifan Ruan, Srinath Sridhar, and Daniel Ritchie. 2022a. Unsupervised Kinematic Motion Detection for Part-Segmented 3D Shape Collections. In *ACM SIGGRAPH 2022 Conference Proceedings* (Vancouver, BC, Canada) *(SIGGRAPH '22)*. Article 2, 9 pages. doi:10.1145/3528233.3530742

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. 2020. RigNet: neural rigging for articulated characters. *ACM Trans. Graph.* 39, 4, Article 58 (Aug. 2020), 14 pages. doi:10.1145/3386569.3392379

Zhan Xu, Yang Zhou, Li Yi, and Evangelos Kalogerakis. 2022b. MoRig: Motion-Aware Rigging of Character Meshes from Point Clouds. In *SIGGRAPH Asia 2022 Conference Papers* (Daegu, Republic of Korea) *(SA '22)*. Article 1, 9 pages. doi:10.1145/3550469.3555390

Zihao Yan, Ruizhen Hu, Xingguang Yan, Luanmin Chen, Oliver Van Kaick, Hao Zhang, and Hui Huang. 2019. RPM-Net: recurrent prediction of motion and parts from point cloud. *ACM Trans. Graph.* 38, 6, Article 240 (Nov. 2019), 15 pages. doi:10.1145/3355089.3356573

Li Yi, Haibin Huang, Difan Liu, Evangelos Kalogerakis, Hao Su, and Leonidas Guibas. 2018. Deep part induction from articulated object pairs. *ACM Transactions on Graphics* 37, 6 (Dec. 2018), 1–15. doi:10.1145/3272127.3275027

Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. 2016. A scalable active framework for region annotation in 3D shape collections. *ACM Trans. Graph.* 35, 6, Article 210 (Dec. 2016), 12 pages. doi:10.1145/2980179.2980238

Ran Zhang, Thomas Auzinger, and Bernd Bickel. 2021. Computational Design of Planar Multistable Compliant Structures. *ACM Trans. Graph.* 40, 5, Article 186 (Oct. 2021), 16 pages. doi:10.1145/3453477

Yuchen Zhou, Jiayuan Gu, Tung Yen Chiang, Fanbo Xiang, and Hao Su. 2024. Point-SAM: Promptable 3D Segmentation Model for Point Clouds. arXiv:2406.17741 [cs.CV]

Yuchen Zhou, Jiayuan Gu, Xuanlin Li, Minghua Liu, Yunhao Fang, and Hao Su. 2023. PartSLIP++: Enhancing Low-Shot 3D Part Segmentation via Multi-View Instance Segmentation and Maximum Likelihood Estimation. arXiv:2312.03015 [cs.CV]

A 3D Segmentation Hyperparameters

In Tables 3 through 8, we compile a comprehensive comparison of the different hyper-parameters that can be chosen when obtaining the 2D segmentations. We considered 4 parameters: the number of images that we were to pass to GLIP and SAM; the resolution of each image; the confidence cutoff for GLIP to accept the bounding box that it found; and the type of prompt that we input to GLIP, where *Word* stands for the prompt "wheel" while *Phrase* refers to the prompt "wheel of a vehicle". We obtained these results for 15 different vehicles of varying topology and vertex count, and compared them to a ground truth segmentation. The first two columns of data relate to the area that was labeled as a wheel due to the image segmentation: we find that at most 83% of the surface area that makes up a wheel is correctly identified, and this is only achieved in the cases where a large amount of images are captured. However, these same hyperparameter configurations also tend to misidentify a larger surface area of the chassis as a wheel, and as such are undesirable. This clearly demonstrates the need to optimize the size and shape of the segments to capture the missing data. The last two columns report how many wheel segments were created by DBSCAN. Although none of the configurations were able to create the exact number of segments required, we found that in all cases where exactly one vehicle obtained less wheels than expected, it was in the case of the CEMENT MIXER; it has 4 pairs of co-axial wheels that are very close together, and thus DBSCAN creates 4 segments for them instead of 8. Although this is still a failure case, this is less serious than non-coaxial wheels being joined as one, showing that this can be a good technique for separating the 3D hit points that we obtain from the image segmentations.

The data is split between six tables due to its volume, but it should be considered as one unit. Across the tables, we follow the convention that a darker colour is equivalent to a more desirable result. The parameter combination that we end up using for the paper can be found emphasized in Table 4.

Table 3. Evaluation of the accuracy of 3D segmentation using the image segmentation projected on the models, across a variety of hyperparameters. Here, 4 images are used.

Hyperparameters			Area Captured		Number of wheels created	
Image res	Cutoff	Prompt type	True Positive	False Positive	Same as GT	More than GT
350 px	50%	Word	38.3%	1.0%	53.3%	40.0%
		Phrase	37.8%	1.0%	66.7%	26.7%
	60%	Word	35.5%	1.0%	60.0%	26.7%
		Phrase	35.7%	1.0%	60.0%	26.7%
	75%	Word	19.5%	0.8%	13.3%	20.0%
		Phrase	22.3%	0.6%	40.0%	20.0%
700 px	50%	Word	53.6%	1.7%	73.3%	20.0%
		Phrase	53.2%	1.4%	80.0%	13.3%
	60%	Word	47.9%	1.2%	86.7%	6.7%
		Phrase	49.1%	1.3%	80.0%	6.7%
	75%	Word	25.8%	0.7%	26.7%	13.3%
		Phrase	26.2%	0.6%	20.0%	20.0%
1080 px	50%	Word	57.3%	1.5%	73.3%	26.7%
		Phrase	56.4%	1.4%	73.3%	26.7%
	60%	Word	53.2%	1.4%	60.0%	33.3%
		Phrase	52.0%	1.2%	46.7%	46.7%
	75%	Word	24.0%	0.9%	33.3%	6.7%
		Phrase	12.2%	0.0%	26.7%	0.0%

Table 4. Results when 10 images are used.

Hyperparameters			Area Captured		Number of wheels created	
Image res	Cutoff	Prompt type	True Positive	False Positive	Same as GT	More than GT
350 px	50%	Word	61.1%	3.2%	80.0%	13.3%
		Phrase	61.1%	2.7%	86.7%	6.7%
	60%	Word	58.9%	2.5%	86.7%	6.7%
		Phrase	59.4%	2.5%	93.3%	0.0%
	75%	Word	37.7%	1.6%	60.0%	13.3%
		Phrase	43.7%	1.8%	73.3%	6.7%
700 px	**50%**	Word	70.3%	3.4%	86.7%	6.7%
		Phrase	70.2%	3.3%	93.3%	0.0%
	60%	Word	69.2%	3.1%	93.3%	0.0%
		Phrase	69.4%	3.1%	93.3%	0.0%
	75%	Word	47.9%	2.0%	73.3%	0.0%
		Phrase	48.7%	1.8%	80.0%	0.0%
1080 px	50%	Word	72.6%	3.5%	73.3%	13.3%
		Phrase	72.3%	3.2%	80.0%	6.7%
	60%	Word	71.7%	2.9%	86.7%	6.7%
		Phrase	71.3%	3.0%	86.7%	6.7%
	75%	Word	41.3%	2.0%	60.0%	6.7%
		Phrase	35.2%	1.3%	60.0%	6.7%

Table 5. Results when 16 images are used.

Hyperparameters			Area Captured		Number of wheels created	
Image res	Cutoff	Prompt type	True Positive	False Positive	Same as GT	More than GT
350 px	50%	Word	65.3%	4.2%	86.7%	6.7%
		Phrase	65.5%	3.9%	80.0%	6.7%
	60%	Word	63.5%	3.5%	86.7%	6.7%
		Phrase	64.0%	3.4%	93.3%	0.0%
	75%	Word	43.1%	2.1%	66.7%	6.7%
		Phrase	46.7%	2.3%	66.7%	0.0%
700 px	50%	Word	73.3%	4.0%	80.0%	6.7%
		Phrase	73.2%	3.7%	80.0%	6.7%
	60%	Word	71.9%	3.3%	80.0%	6.7%
		Phrase	72.2%	3.2%	86.7%	0.0%
	75%	Word	50.7%	2.2%	73.3%	0.0%
		Phrase	55.4%	1.9%	86.7%	0.0%
1080 px	50%	Word	75.3%	4.1%	73.3%	6.7%
		Phrase	75.1%	4.0%	80.0%	6.7%
	60%	Word	74.0%	3.6%	73.3%	6.7%
		Phrase	74.0%	3.3%	80.0%	6.7%
	75%	Word	46.0%	1.9%	80.0%	0.0%
		Phrase	38.4%	1.3%	66.7%	0.0%

Table 6. Evaluation of the accuracy of 3D segmentation using the image segmentation projected on the models, across a variety of hyperparameters, where half of the images were taken with a view of the vehicles from underneath, and half were taken with the standard view. Here, 8 images are used.

Hyperparameters			Area Captured		Number of wheels created	
Image res	Cutoff	Prompt type	True Positive	False Positive	Same as GT	More than GT
350 px	50%	Word	57.0%	2.4%	86.7%	13.3%
		Phrase	57.3%	1.9%	86.7%	13.3%
	60%	Word	50.8%	1.8%	60.0%	26.7%
		Phrase	54.5%	1.8%	80.0%	20.0%
	75%	Word	27.0%	1.3%	13.3%	33.3%
		Phrase	33.4%	1.1%	33.3%	40.0%
700 px	50%	Word	69.3%	2.7%	86.7%	6.7%
		Phrase	69.3%	2.0%	93.3%	0.0%
	60%	Word	65.8%	1.8%	93.3%	0.0%
		Phrase	66.3%	1.9%	93.3%	0.0%
	75%	Word	37.5%	0.9%	53.3%	6.7%
		Phrase	37.6%	0.8%	53.3%	0.0%
1080 px	50%	Word	72.5%	3.3%	86.7%	6.7%
		Phrase	71.9%	2.8%	93.3%	0.0%
	60%	Word	68.3%	2.7%	86.7%	0.0%
		Phrase	68.6%	2.3%	86.7%	6.7%
	75%	Word	32.3%	1.2%	46.7%	6.7%
		Phrase	22.0%	0.4%	26.7%	13.3%

Table 7. Results when 20 images are used

Hyperparameters			Area Captured		Number of wheels created	
Image res	Cutoff	Prompt type	True Positive	False Positive	Same as GT	More than GT
350 px	50%	Word	74.1%	5.5%	80.0%	6.7%
		Phrase	74.2%	4.8%	80.0%	6.7%
	60%	Word	72.3%	4.3%	93.3%	0.0%
		Phrase	73.1%	4.4%	93.3%	0.0%
	75%	Word	51.0%	3.1%	66.7%	6.7%
		Phrase	57.7%	3.3%	80.0%	0.0%
700 px	50%	Word	80.6%	6.4%	73.3%	13.3%
		Phrase	80.5%	6.2%	73.3%	13.3%
	60%	Word	80.1%	5.4%	80.0%	6.7%
		Phrase	80.1%	5.7%	80.0%	6.7%
	75%	Word	57.1%	3.6%	73.3%	6.7%
		Phrase	62.1%	3.9%	66.7%	13.3%
1080 px	50%	Word	82.1%	7.8%	73.3%	13.3%
		Phrase	82.0%	7.0%	73.3%	13.3%
	60%	Word	81.5%	6.5%	73.3%	13.3%
		Phrase	81.2%	6.6%	73.3%	13.3%
	75%	Word	51.1%	3.0%	66.7%	13.3%
		Phrase	44.6%	2.3%	60.0%	13.3%

Table 8. Results when 32 images are used

Hyperparameters			Area Captured		Number of wheels created	
Image res	Cutoff	Prompt type	True Positive	False Positive	Same as GT	More than GT
350 px	50%	Word	76.8%	8.0%	80.0%	6.7%
		Phrase	77.0%	7.7%	73.3%	13.3%
	60%	Word	75.4%	6.9%	80.0%	6.7%
		Phrase	76.0%	6.8%	86.7%	0.0%
	75%	Word	54.4%	3.8%	80.0%	0.0%
		Phrase	60.4%	4.3%	73.3%	0.0%
700 px	50%	Word	82.2%	10.5%	86.7%	0.0%
		Phrase	82.1%	9.9%	80.0%	6.7%
	60%	Word	81.6%	8.9%	80.0%	6.7%
		Phrase	81.6%	8.6%	86.7%	0.0%
	75%	Word	62.1%	5.9%	86.7%	0.0%
		Phrase	68.4%	4.7%	86.7%	0.0%
1080 px	50%	Word	83.4%	11.1%	73.3%	0.0%
		Phrase	83.4%	9.5%	73.3%	6.7%
	60%	Word	82.8%	9.0%	73.3%	6.7%
		Phrase	82.9%	8.6%	80.0%	6.7%
	75%	Word	55.7%	4.9%	80.0%	0.0%
		Phrase	52.5%	3.1%	73.3%	0.0%