OpenHullGAN: A Rapid 3D Ship-Hull Generation Cookbook

Kirsten Odendaal
MARIN
Wageningen, NL
k.odendaal@marin.nl

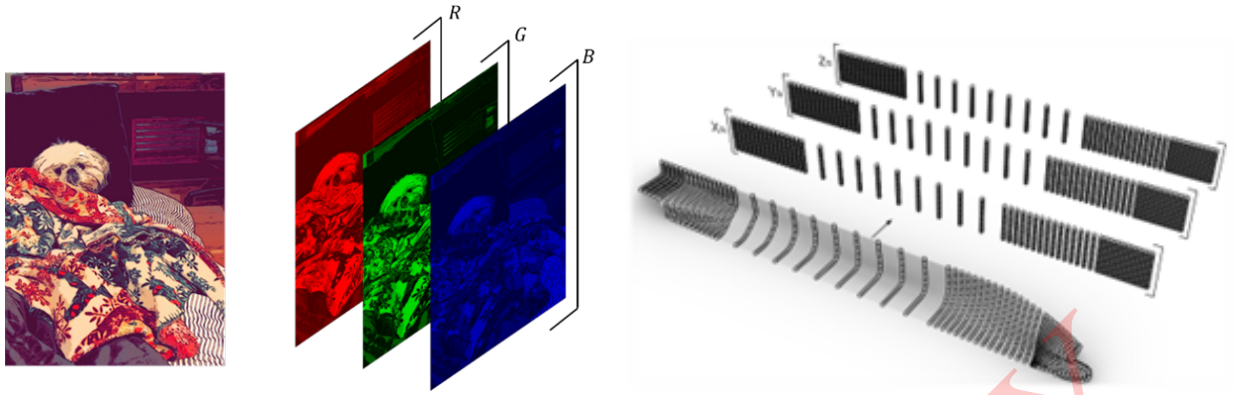
Abstract

Past work on *ShipHullGAN* demonstrated that generative adversarial networks (GANs) can create plausible 3-D ship-hull forms, but key implementation details were omitted, hindering adoption and independent verification. We present OpenHull-GAN, a fully documented Deep Convolutional GAN (DCGAN) workflow to automatically produce diverse, high-fidelity 3D ship hull forms. We curate a dataset of 12,000 hull geometries spanning six distinct vessel classes (e.g., sailing yachts, planing craft, motor yachts, etc.), and encode each hull's surface into a structured 3D representation amenable to convolutional neural networks. The proposed DCGAN architecture employs novel loss functions – a space-filling loss to encourage output diversity and a Laplace loss to enforce surface smoothness – alongside the standard adversarial objective. After training, the generator network produces realistic hull geometries that capture key design features and variations across vessel types. Quantitative evaluation and visual inspections demonstrate that the GAN learns meaningful latent design features, enabling controlled hull form interpolation and exploration. Key results include successful generation of plausible hull forms with systematically varying characteristics and the ability to discover novel designs outside the original dataset. We discuss the challenges encountered, such as mode collapse and geometric artifacts, and how techniques like latent space regularization, post-processing (Gaussian smoothing, symmetry enforcement), and small-batch training stabilized learning. This DCGAN-based approach highlights the potential of generative models to revolutionize ship design by facilitating rapid, free-form hull generation with greater diversity and efficiency than conventional methods. Finally, we outline avenues for future improvements, including expanded training data, advanced GAN architectures, and integration of physics-based performance metrics to guide the generation process. All source code, processed data, and YAML configuration files are released under the MIT licence, enabling researchers and practitioners to replicate results in ≤ 2 hours of GPU time, swap in alternative losses, or port the pipeline to new vessel classes. By turning an opaque proof-of-concept into an open cookbook, OpenHull-GAN lowers the barrier for applying GANs in naval design workflows and provides a scaffold for future physics-informed or conditional extensions.

Keywords First keyword · Second keyword · More

1 Introduction

Ship designers seek to explore a vast space of hull shapes to identify innovative, high-performance designs. The ability to freely form and deform hull geometries can significantly expand the design space, uncovering novel solutions that meet stringent hydrodynamic and structural requirements. However, traditional hull design methodologies struggle to balance accuracy, diversity, and computational efficiency. Iterative manual design or parametric modeling can be time-consuming and may not capture the full spectrum of feasible shapes, especially for complex features like bulbous bows or transom sterns. There is a growing need for automated approaches that can rapidly generate high-quality hull forms while preserving essential geometric fidelity and diversity of designs.



(a) RGB (b) XYZ

Figure 1: Relating general image structures to 3D geometries. (a) A conventional image can be represented in RGB format, which is a type of numerical embedding in which a pixel ranges between 0-255. (b) The equivalent 3D representation where XYZ coordinates are used instead

Exploring that space quickly and reproducibly is still an open challenge for both naval architects and machine-learning researchers. Generative Adversarial Networks (GANs) [1] have revolutionised 2-D image synthesis and recently begun to influence engineering design. PaDGAN demonstrated diversity-aware part generation [2], and ShipHullGAN showed that a Deep Convolutional GAN (DCGAN) [3] can yield plausible 3-D hull forms. Yet independent adoption has stalled: ShipHullGAN's training scripts, hyper-parameters and dataset remain proprietary, impeding verification and extension. Parallel work with diffusion models (ShipGen [4] and the conditional C-ShipGen [5]) highlights the community's appetite for transparent baselines. This paper closes that reproducibility gap. We present OpenHullGAN, an end-to-end, open-source DCGAN workflow and dataset for rapid 3-D ship-hull generation. The technical idea is to recast each hull's surface as a 2-D lattice where the (x, y, z) coordinates are stored in three channels, exactly like the RGB channels of an image. This encoding (i) preserves local surface topology, (ii) fits naturally into mature 2-D convolutional libraries, and (iii) avoids the memory footprint of voxel grids or signed-distance fields. Figure 1 illustrates this concept, relating a conventional image representation to the 3D hull surface grid representation. Using this approach, the generator network learns a latent design space where each random input vector corresponds to a distinct hull geometry. By navigating this latent space, designers can explore smooth transformations between hull forms, enabling parametric design morphing and optimization directly through the learned features of the GAN. Building on that representation, we:

1. Release full source code, YAML configs and the processed dataset (MIT licence) so any practitioner can reproduce results in ≤ 2 GPU-hours.
2. Detail a robust, space-filling Latin-Hypercube pipeline that parameterises 12,000 hulls across six vessel classes and exports them as aligned $20 \times 40 \times 3$ tensors.
3. Specify a baseline DCGAN architecture and training recipe, adding two domain-specific losses: (i) a space-filling diversity term to mitigate mode collapse, and (ii) a Laplace smoothness term that regularises local curvature.
4. Evaluate the generator qualitatively with domain experts and quantitatively via hull-shape embeddings, showing diversity comparable to the real dataset and fidelity superior to a PCA baseline.

By turning a previously opaque proof-of-concept into a cookbook, OpenHullGAN lowers the barrier for applying GANs to naval design and supplies a scaffold for future advances, such as conditional diffusion models or wavelet-latent architectures [6]. The rest of the paper proceeds as follows: Section 2 explains data encoding and GAN training; Section 3 analyses results; Section 4 discusses limitations and applications; Section 5 outlines future work; and Section 6 concludes.

2 Methodology

The agent processes sensory inputs using a multi-modal network (summarized in Table **??**).

Figure Placeholder: Workflow Diagram

Figure 2: Complete GAN process flow diagram.

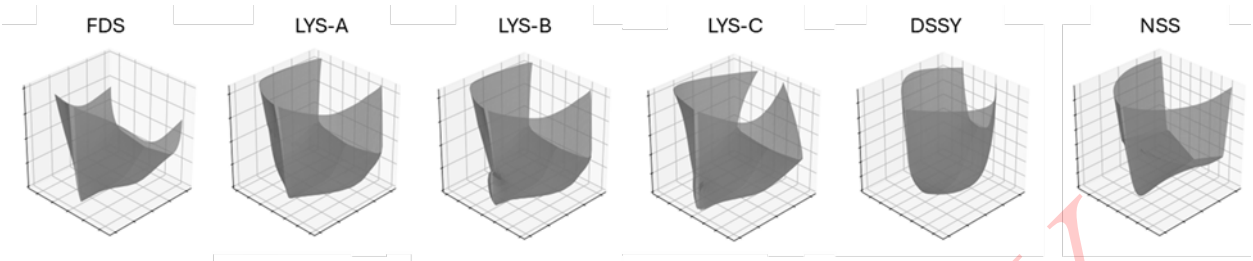


Figure 3: Normalized parent hull geometries.

2.1 Data Collection and Geometry Representation

We compiled a corpus of 12,000 parametric hulls spanning six vessel classes; sailing yachts, high-speed planing craft, and various motor yachts. Normalized representations of these parent geometries can be seen visualized in Figure 3. Each hull is generated by a Rhino/Grasshopper script that exposes both global parameters (length-to-beam ratio, prismatic coefficient) and local shape controls (bulb radius, transom flare, etc.). Parameter values are drawn with Latin Hypercube Sampling (LHS) [7] to achieve a space-filling distribution and prevent over-representation of any design sub-region. The Grasshopper routine exports every hull as a watertight NURBS surface (native .3dm). We then sample the starboard half onto a fixed 20×40 lattice defined by:

- $s \in [0, 1]$: normalized station measured aft (s=0) to fwd perpendicular (s=1),
- $v \in [0, 1]$: normalized vertical position measured keel (v=0) to deck (v=1). The query (s,v) is converted to (x, y, z) by surface evaluation, yielding a tensor of shape $20 \times 40 \times 3$.

Figure 4 visualises the sampling grid and its correspondence to the 3-channel "image" fed to the DCGAN. The general process was adoped from [3], however alternative representations can be used and explored. Training on the starboard half (mirror-symmetric about the y=0 centre-plane) halves memory cost and implicitly teaches the network global symmetry; the port side can be reconstructed by simple mirroring at inference time. Before sampling, every hull is rigid-aligned and scaled to a unit bounding box:

- $x \in [0, 1]$ covers length between perpendiculars with x=0 at the transom plane,
- $y \in [0, 0.5]$ covers half-beam to enforce centre-plane symmetry,
- $z \in [0, 1]$ spans keel to deck (positive upward).

This normalisation removes scale bias and lets the GAN focus on shape variation. Each tensor is saved as a compressed NumPy .npz file (float32, $\approx$ 9 kB) and accompanied by a human-readable .csv for rapid inspection. Overall, the procedure converts heterogeneous CAD surfaces into a homogeneous, GPU-friendly representation without sacrificing geometric fidelity.

2.2 GAN Architecture

We implemented a Deep Convolutional GAN (DCGAN) architecture [8], realized using the PyTorch framework, to serve as the generative model. The architecture comprises two core neural networks: a Generator (G) and a Discriminator (D), both operating on the $3 \times 20 \times 40$ (Channels × Height × Width) tensor representation described previously. The detailed layer structure for both networks is presented in Table 1. The Generator network G maps a latent vector z (a $d_z = 25$ dimensional vector sampled from a standard normal distribution $N(0, I)$) to a $3 \times 20 \times 40$ tensor representing a normalized hull point cloud. The latent vector is first reshaped to $(25, 1, 1)$ and fed into a series of five ConvTranspose2d layers that progressively upsample the spatial dimensions (from 3×3 initially, up to the target 20×40) while adjusting the number of feature channels (starting from $ngf \times 8 = 512$ down to $nc = 3$, where $ngf = 64$). Batch Normalization is applied after each transposed convolution (except the output layer) to stabilize training [9], followed by ReLU activation. The final layer uses a Tanh activation function, producing outputs in the range [-1, 1]. These outputs are subsequently rescaled to the original [0, 1] normalized coordinate range.

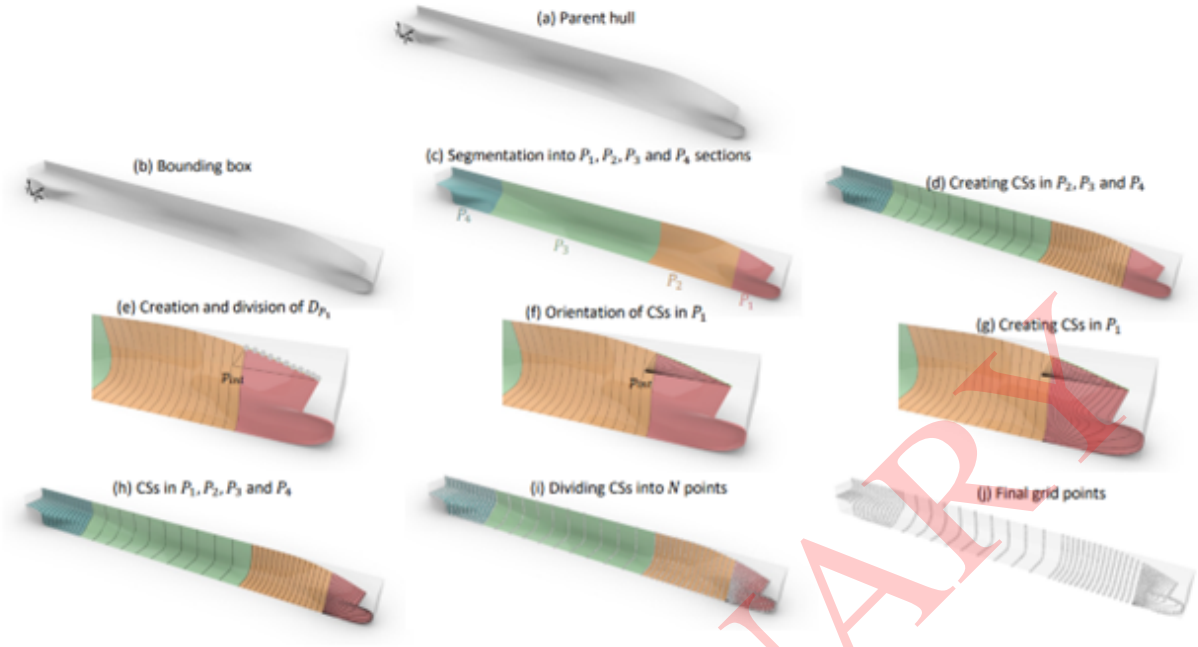


Figure 4: Generalised encoding strategy employed to convert 3D geometries into appropriate embedding structures. [3]

Table 1: Detailed layer specifications for the Generator (G) and Discriminator (D) networks used in OpenHullGAN where $ndf = ngf = 64$.

Generator (G)

Layer	Type	Out Channels	Kernel Size	Stride	Padding	Batch Norm	Activation	Output Shape (C, H, W)
Input	-	25	-	-	-	-	-	(25, 1, 1)
G-TC1	ConvTranspose2d	$ngf \times 8 = 512$	(3, 3)	(1, 1)	(0, 0)	Yes	ReLU	(512, 3, 3)
G-TC2	ConvTranspose2d	$ngf \times 4 = 256$	(3, 3)	(2, 2)	(1, 1)	Yes	ReLU	(256, 5, 5)
G-TC3	ConvTranspose2d	$ngf \times 2 = 128$	(4, 3)	(2, 1)	(1, 1)	Yes	ReLU	(128, 10, 5)
G-TC4	ConvTranspose2d	$ngf = 64$	(4, 4)	(2, 2)	(1, 1)	Yes	ReLU	(64, 20, 10)
G-TC5	ConvTranspose2d	$nc = 3$	(4, 4)	(2, 2)	(1, 1)	No	Tanh	(3, 20, 40)

Discriminator (D)

Layer	Type	Out Channels	Kernel Size	Stride	Padding	Batch Norm	Activation	Output Shape (C, H, W)
Input	-	3	-	-	-	-	-	(3, 20, 40)
D-Pre	GaussianNoise+Dropout(0.4)	3	-	-	-	-	-	(3, 20, 40)
D-C1	Conv2d	$ndf = 64$	(4, 4)	(2, 2)	(1, 1)	No	Leaky ReLU (0.2)	(64, 10, 20)
D-C2	Conv2d	$ndf \times 2 = 128$	(4, 4)	(2, 2)	(1, 1)	Yes	Leaky ReLU (0.2)	(128, 5, 10)
D-C3	Conv2d	$ndf \times 4 = 256$	(4, 3)	(2, 1)	(1, 1)	Yes	Leaky ReLU (0.2)	(256, 3, 10)
D-C4	Conv2d	$ndf \times 8 = 512$	(3, 3)	(2, 2)	(1, 1)	Yes	Leaky ReLU (0.2)	(512, 2, 5)
D-C5	Conv2d	1	(2, 5)	(1, 1)	(0, 0)	No	Sigmoid	(1, 1, 1)

The Discriminator network D takes a $3 \times 20 \times 40$ tensor (either real or fake) and outputs a scalar probability indicating whether the input is real. Before the first convolution, Gaussian noise is added to the input, and a small dropout (40%) is applied for regularization. D then employs five Conv2d layers with strides to progressively downsample the spatial resolution (from 20×40 down to 1×1) and increase feature channels (from $nc = 3$ up to $ndf * 8 = 512$, where $ndf = 64$). Leaky ReLU activation (with a negative slope of 0.2) is used after each convolution (except the last). Batch Normalization is applied in intermediate layers. The final Conv2d layer reduces the output to a single channel with spatial dimensions 1x1, which is then passed through a Sigmoid activation to produce the probability output. We also experimented with features like Spectral Normalization [10] and additional dropout layers, but found this configuration provided stable results and the added value was not explicitly clear. Both networks were initialized with random weights. We used the Adam optimizer with a learning rate of 2×10^{-4} and beta parameters ($\beta_1 = 0.5$, $\beta_2 = 0.999$) for both G and D [8]. Stabilization techniques are discussed further in Section 2.3.

2.3 Training Procedure and Loss Functions VERSION 1

Training the DCGAN follows the standard adversarial minimax framework [1]. We iteratively update the Discriminator D and Generator G. The objective for D is to maximize its ability to distinguish real hulls x from generated hulls $G(z)$, using the binary cross-entropy loss:

$$L_D = -E_{x\sim p_{data}}[\log D_{\theta,D}(x)] - E_{z\sim p_z}[\log(1 - D_{\theta,D}(G_{\theta,G}(z)))] \tag{1}$$

represent the network parameters. The objective for G is to minimize the probability of its outputs being classified as fake. We use the non-saturating adversarial loss for G:

$$L_G = -E_{z\sim p_z}[\log D_{\theta,D}(G_{\theta,G}(z))] \tag{2}$$

During each training iteration, we first update D's parameters θ_D using a batch (size 128) of real hulls and a batch of fake hulls from the current G. Then, we update G's parameters θ_G using a fresh batch of latent vectors z. We trained for 500 epochs, monitoring loss curves and output quality. While L_G drives realism, we found it essential to add regularization terms to improve diversity and surface quality based on insights from prior work [2, 3].

1. Space-Filling Diversity Loss (L_{div}): To prevent mode collapse and encourage G to explore the entire design space, we added a diversity loss based on pairwise distances between generated samples within a batch. It penalizes low dispersion in the output coordinate space:

$$L_{div} = \sum_{i<j} \exp\big(-\|G(z_i) - G(z_j)\|^2\big), \tag{3}$$

   where the sum is over distinct pairs (i, j) in a batch, and $||\cdot||_2^2$ is the squared Euclidean distance between the flattened $20 \times 40 \times 3$ output tensors G(z).

2. Laplace Smoothness Loss (L_{lap}): To reduce high-frequency noise and encourage smoother surfaces, we incorporated a Laplace loss based on the L1 norm of the discrete Laplacian operator applied to the generated point grid $X = G_{\theta_G}(z)$:

$$L_{\text{lap}} = \sum_i \left\|\Delta x_i\right\|_1, \tag{4}$$

   Here, N=20, M=40, and $X_{i,j}$ is the (x, y, z) coordinate vector at grid position (i, j). The discrete Laplacian $\Delta X_{i,j}$ is computed using 4-connectivity (summing vector differences between $X_{i,j}$ and its four adjacent neighbors on the grid):

$$\Delta X_{i,j} = \sum_{(i',j')\in N(i)} (x_j - x_i) \tag{5}$$

   Boundary points are handled using reflection padding (Neumann boundary conditions). Minimizing L_{lap} encourages local surface smoothness.

The final generator loss combines these terms with annealing weights $\Gamma(t)$ and λ(t): $L_{G,final}(t) = L_{G,adv} + \Gamma(t) \cdot L_{div} + \lambda(t) \cdot L_{lap}$ The weights increase over training epochs t from 0 up to final values Γ_0 and λ_0 using the schedule $\Gamma(t) = \Gamma_0(t/T)^p$ (and similarly for λ), where $T = 500$ is the total epochs. We used p=2 (quadratic ramp-up) and set the final weights based on default values common in related generative modeling literature, using $\Gamma_0 = [0.1]$ and $\lambda_0 = [0.1]$, without extensive hyperparameter search for these specific weights in this work.

To further stabilize training, we employed:

1. Label smoothing for the discriminator (labeling real samples as 0.9 instead of 1.0),
2. Gaussian noise injection to the discriminator's input (as implemented in the D architecture, Sec 2.2).

We experimented with Spectral Normalization [10] but did not include it in the final model configuration presented here. Training continued until the generator produced diverse outputs (assessed visually and via loss monitoring) and the geometry reconstruction (indicating successful conversion to smooth CAD surfaces via Sec 2.4 post-processing) was deemed satisfactory.

2.4 Training Procedure and Loss Functions VERSION 2

Training the DCGAN involves iteratively updating D and G to play a minimax game. The adversarial loss driving this game is based on binary cross-entropy. In its original form [1], the discriminator's loss L_D and generator's loss L_G can be written as:

$$L_D = -E_{x\sim p_{data}}[\log D(x)] - E_{z\sim p_z}[\log(1 - D(G(z)))] \tag{6}$$

$$L_G = -E_{z\sim p_z}[\log D(G(z))] \tag{7}$$

where $D(x)$ is the discriminator's estimated probability that sample x is real. In practice, we use the non-saturating heuristic for the generator loss (maximizing $\log D(G(z))$). During each training iteration, we perform two alternating steps:

1. Update D using a batch of real hull samples and an equal-sized batch of generated samples (from the current G) to maximize L_D (make D better at discrimination).
2. Update G using a fresh batch of latent vectors to minimize L_G (make G produce more realistic outputs that D would classify as real).

This two-step update constitutes one training epoch. We trained the GAN for many epochs, monitoring the losses and the visual quality of generated hulls to select the final model. Early in training, D quickly learns to differentiate real vs. fake, and G outputs are often noisy or unrealistic. As training progresses, G improves and the two networks reach an approximate equilibrium where generated hulls appear increasingly plausible.

While the pure adversarial loss can in theory drive G to produce realistic outputs, in practice we found it beneficial to introduce additional loss terms to address specific challenges in hull generation. Two custom regularization losses were added to the generator's objective: a space-filling loss and a Laplace smoothness loss. The space-filling loss L_{div} is designed to encourage diversity in the generated samples by penalizing the generator if outputs start collapsing to similar shapes. We implement L_{div} following the approach Chen and Ahmed [2]: it measures the dispersion of a batch of generated designs in the latent or feature space and penalizes low dispersion. One formulation is to use pairwise distances between generated samples to construct a space-filling criterion S; for example,

$$L_{div} = \sum_{i<j} \exp\big(-\|G(z_i) - G(z_j)\|^2\big), \tag{8}$$

which is small when samples $G(z_i)$ are far apart (desirable) and larger when samples are similar. The generator is then tasked to minimize L_{div} along with the adversarial loss. In practice, we incorporate this via a weighted term $\Gamma\, L_{div}$ added to the loss. Thus, the augmented generator objective becomes:

$$L'_G = L_G + \Gamma\, \mathbb{L}_{div}, \tag{9}$$

where Γ is a weight that controls the importance of the diversity term. We schedule Γ to increase over the course of training: initially $\Gamma = 0$ (focus on realism first) and gradually ramp it up to a final value Γ_0 as training progresses (to promote diversity once the generator has learned basic realism). We use a schedule:

$$\Gamma(t) = \Gamma_0 \left(\frac{t}{T}\right)^p \tag{10}$$

as suggested by Chen and Ahmed [2], where t is the current epoch and T is the total number of epochs, with p controlling how nonlinearly Γ grows. This annealing strategy ensures that the diversity penalty does not hinder early training, while later epochs strongly encourage exploring the full output space.

The Laplace smoothness loss L_{lap} is introduced to reduce high-frequency noise and enforce smoother surfaces in generated hulls. Because the GAN operates on a grid of points, it might produce slight oscillations or unevenness in the hull surface if not guided. The Laplace loss leverages the discrete Laplacian operator on the 2D grid of points. For each point i in the generated hull grid, we compute the Laplacian Δx_i as the sum of differences between that point's coordinate and its neighbors j in the grid:

$$\Delta x_i = \sum_{j\in N(i)} (x_j - x_i) \tag{11}$$

where $N(i)$ denotes the neighboring points of i. This yields a measure of local curvature or variation. The Laplace loss is then defined as the sum of absolute Laplacians over all points:

$$L_{\text{lap}} = \sum_i \left\|\Delta x_i\right\|_1, \tag{12}$$

i.e., the L^1 norm of second-order differences across the surface. By minimizing L_{lap}, the generator is encouraged to produce surfaces where $\Delta x_i \approx 0$ for all points, implying a smoothly varying hull with minimal noise or rippling. Similar to the diversity term, we apply this loss with a weight λ that increases over time. The generator's final loss function combines all terms:

$$L_G^{(\text{total})} = L_G \;+\; \Gamma(t)\, L_{div} \;+\; \lambda(t)\, L_{\text{lap}}\,, \tag{13}$$

with Γ and λ following scheduled increases during training. By the end of training, these additional losses help ensure that the output hulls are not only realistic (thanks to the adversarial loss) but also well-spaced in design space (thanks to L_{div}) and smooth in shape (thanks to L_{lap}).

Throughout training, we monitored the loss curves for signs of instability (such as sudden drops in D loss indicating mode collapse). We employed several stabilization techniques to address GAN training challenges: using relatively small batch sizes (on the order of 128) to reduce gradient noise, applying label smoothing for the discriminator (e.g., label real samples as 0.9 instead of 1.0 to prevent over-confidence), and incorporating a mild Gaussian noise to the inputs of the discriminator to make it less sensitive to minor artifacts in G's output. We also experimented with spectral normalization [10] in the discriminator to constrain its Lipschitz constant, which is known to improve GAN training stability, though our final results were achieved without it. The model was trained on a high-performance GPU; training continued until the generator outputs showed qualitatively good diversity and the validation of geometry reconstruction (see below) was satisfactory (on the order of a few hundred epochs).

2.5 Post-Processing and Geometry Reconstruction

The raw output of the generator is a point cloud representation of a hull's surface. While the generator is trained to produce outputs that resemble the training point clouds, minor artifacts can still appear. Two common issues observed were:

1. slight asymmetry between port and starboard halves (since we only train on half-hulls, the mirrored half may not perfectly match if the output isn't exactly symmetric), and
2. small surface irregularities or noise, especially near high-curvature regions like the bow or stern where the discrete grid approximation is coarse.

To address these, we implemented a post-processing pipeline:

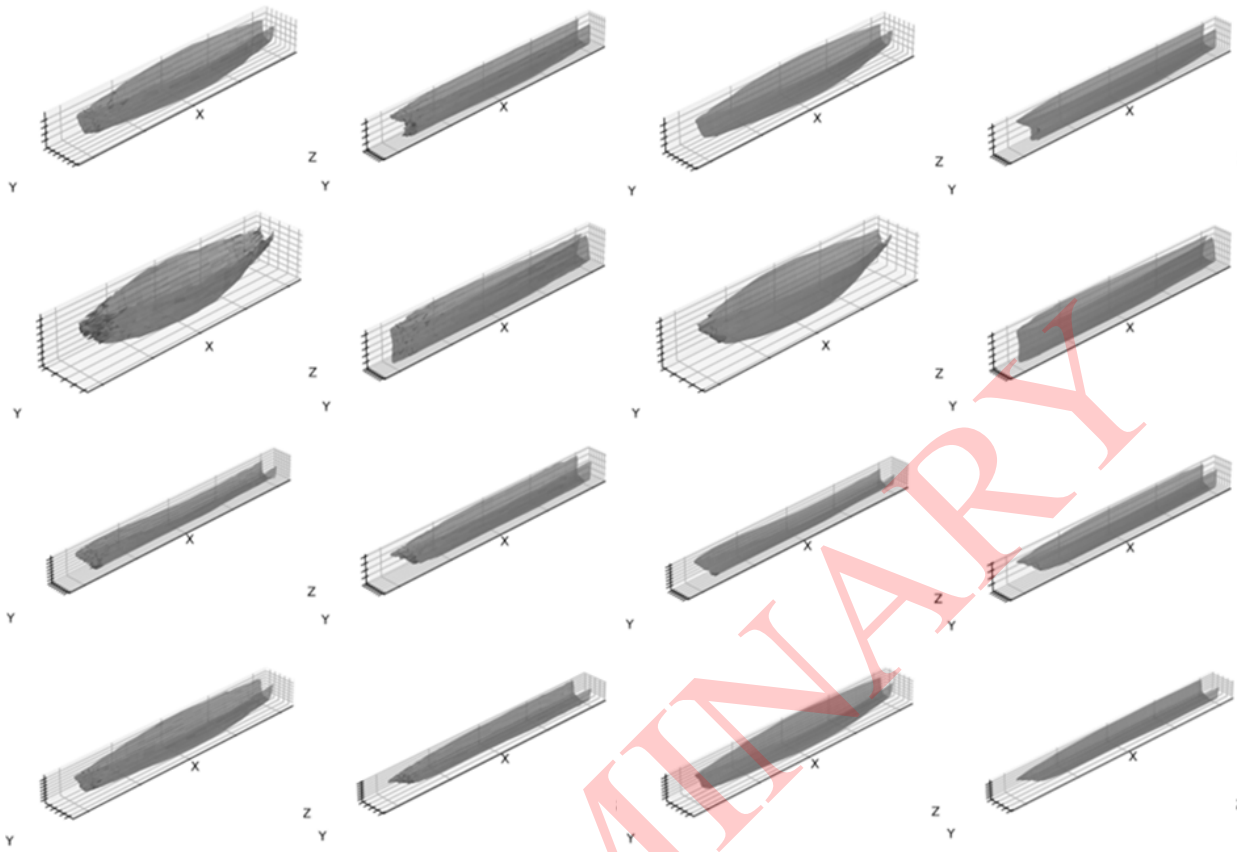
- Symmetry Enforcement: We explicitly mirror the generated half-hull to produce the full hull and then enforce symmetry along the centerline. This is done by averaging any residual differences between the mirrored halves, ensuring the final hull is perfectly symmetric about the longitudinal center plane (a necessary physical constraint for most hull designs). Keel line points are adjusted so that the keel remains straight and centered.
- Surface Smoothing: We apply a light Gaussian smoothing filter to the output point cloud. Essentially, each point's position is adjusted by a weighted average with its neighbours on the grid, which dampens high-frequency noise. Because we incorporated a Laplace loss during training, the generator's output is already reasonably smooth; the Gaussian filter serves as an extra precaution to eliminate any minor bumps or dips without altering the overall hull form significantly. We choose a narrow Gaussian kernel to avoid over-smoothing which could wash out genuine design features.
- Edge Clean-up: For hulls with a transom (flat stern), the bottom edge and transom edge in the point cloud are checked for consistency. If any irregularity is detected (e.g., the generator might produce a slight stagger in what should be a straight line), we correct it by linear interpolation between the nearest stable points on that edge. Similarly, we ensure that the generated bow (foremost point of the hull) lies on the center plane and that the curvature near the bow is fair (monotonic in slope). These corrections are minor and only applied if the GAN output violates basic expected shape properties.

After these steps, the processed point cloud is converted back into a continuous surface representation. Using the same parametric basis as the data generation, we interpolate a smooth surface (for example, using B-splines or NURBS fitting through the generated points). The result is a 3D geometric model of a ship hull that can be exported to common CAD formats or used for downstream analysis (e.g., computational fluid dynamics simulations). Figure 5 shows an example of a raw GAN-generated hull and the final smoothed surface after post-processing.

3 Results

3.1 Hull Generation Quality and Diversity

The trained DCGAN model is capable of producing a wide variety of hull forms by sampling different latent vectors z. Figure 6 presents a gallery of generated hull geometries, illustrating the range of designs learned by the model. These include slender, deep-keel sailing yacht forms, fuller-bodied displacement hulls, and even planing hull shapes with hard chine features. Notably, the generator can interpolate between different hull types: for instance, by gradually changing the latent vector, we observed a smooth transformation from a V-shaped hull with a pronounced keel to a



(a) Raw (b) Gaussian Smoothed

Figure 5: 16 randomly sampled results (full aspect). (a) Direct outputs from the trained generator (b) Gaussian smoothed designs using (k=3, σ=1.2)

flatter bottom hull typical of planing craft. This indicates that the GAN has learned a continuous latent design space where nearby points in latent space correspond to similar hull shapes, and moving along certain directions in latent space yields interpretable changes (e.g., increasing beam, raising chine height, introducing a bulbous bow, etc.). The ability to smoothly morph between hull forms is a powerful feature for design exploration, allowing naval architects to traverse from one design to another and examine intermediate hulls that might not exist in the original dataset.

To evaluate the realism of generated hulls, we conducted both qualitative and quantitative assessments. Qualitatively, domain experts inspected a sample of generated hulls without knowledge of whether they were real or GAN-generated. The majority of generated hulls were deemed plausible and exhibited coherent geometry – continuous fair surfaces without glaring discontinuities. Many designs closely resembled realistic vessels, even though they were artificially generated. A few outliers displayed minor anomalies, such as slightly irregular bow curvature or an exaggerated feature not typical in real designs; these were generally traceable to regions of the design space that were underrepresented in the training data (for example, very extreme hull proportions). Quantitatively, we compared global shape parameters of generated hulls to those of the training set. Key parameters like length-to-beam ratio, prismatic coefficient, and midsection coefficient for generated hulls fell within the range of the dataset and often close to its distribution mean, indicating the GAN did not generate implausible extremes. We also computed a point-to-point error between randomly chosen generated hulls and the closest matching real hull in the dataset (after suitable alignment). The average error was low (on the order of a few percent of hull dimensions), confirming that for each generated hull, one can find a reasonably similar real hull, and vice versa – the generator is covering the space of realistic designs rather than inventing nonsensical shapes.

To quantify diversity, we examined the spread of generated samples in a feature space. We extracted a lower-dimensional embedding of hull shapes (using techniques like principal component analysis on the hull point clouds or utilizing the discriminator's intermediate feature activations as an embedding). In this feature space, we computed metrics such as the pairwise distance distribution among generated samples, comparing it to that of the real dataset. We found that the

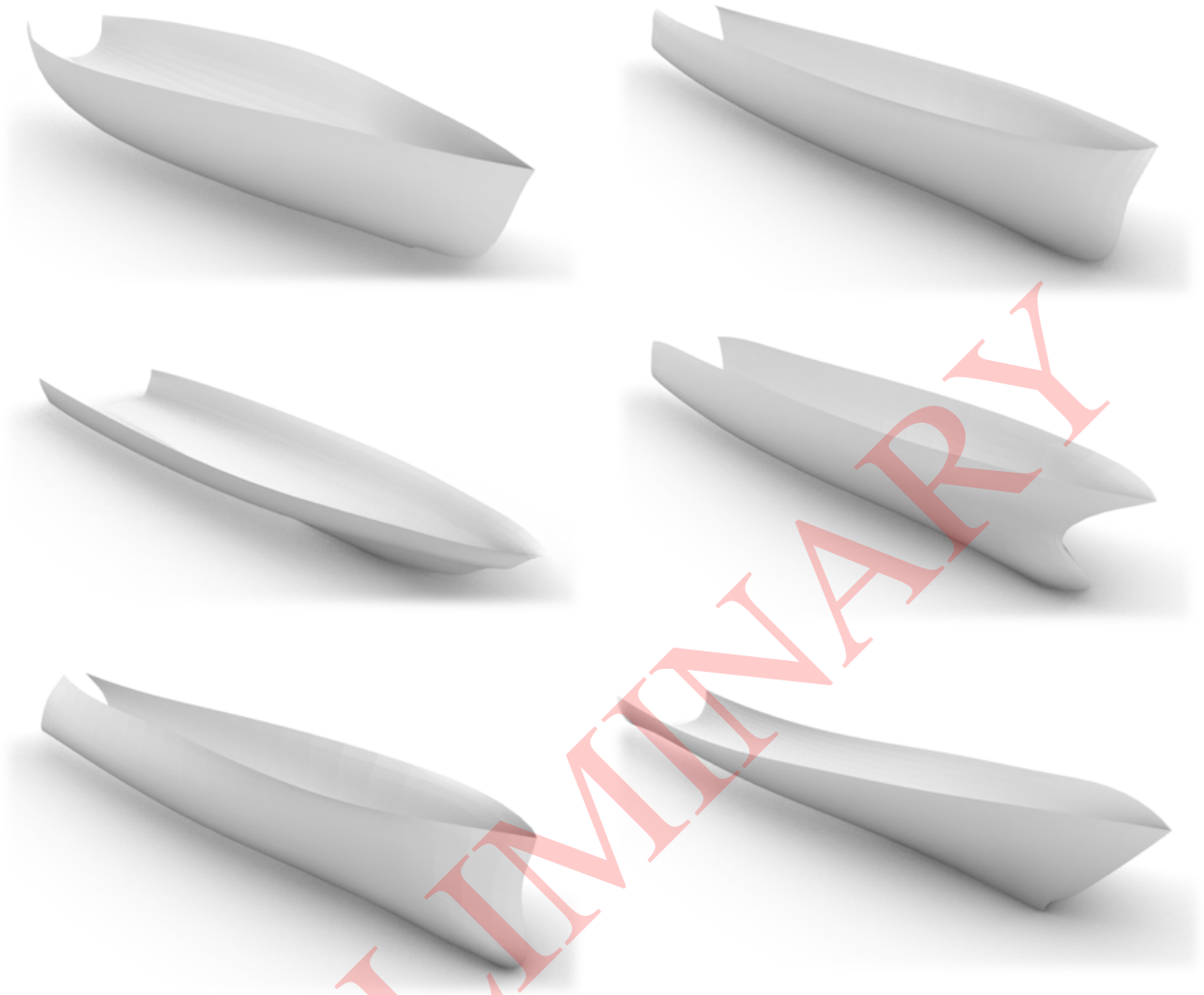


Figure 6: Extracted geometries from a randomly generated sample demonstrating similarities to the parent class geometries.

space-filling loss indeed helped in maintaining a broad coverage: the average pairwise distance among generated hulls was comparable to that of the real hulls, and the diversity did not collapse even after prolonged training. In contrast, a baseline GAN trained without the diversity loss tended to suffer mode collapse after some epochs, producing many near-identical hulls (usually converging to the mean hull shape of the dataset). By the end of training, our model avoided mode collapse – multiple distinct modes of hull designs (corresponding to different ship types) were preserved in the output. For example, clustering analysis on the generated hull set revealed groupings that corresponded to interpretable classes (sailing yacht-like forms clustered separately from motor yacht forms, etc.), showing that the model generates examples from each of the major classes present in training data rather than focusing on just one.

3.2 Reconstruction and Fidelity

The fidelity of generated hulls was further evaluated by reconstructing smooth surfaces from the output point clouds and inspecting critical geometric features. One important test was the forebody reconstruction – the bow region of a ship hull often involves complex curvature (as the surface wraps into a nearly vertical stem). In some early generations, we observed minor artefacts in the forebody, such as unintended wobble in the stem curve or asymmetry in the waterline shape near the bow. After introducing the Laplace smoothness loss and performing post-processing, these artefacts were significantly reduced. The final generated hulls exhibit smooth forebodies that are symmetric and fair, although in a few cases with very narrow bows the fidelity is slightly less than a comparable real design (likely due to fewer

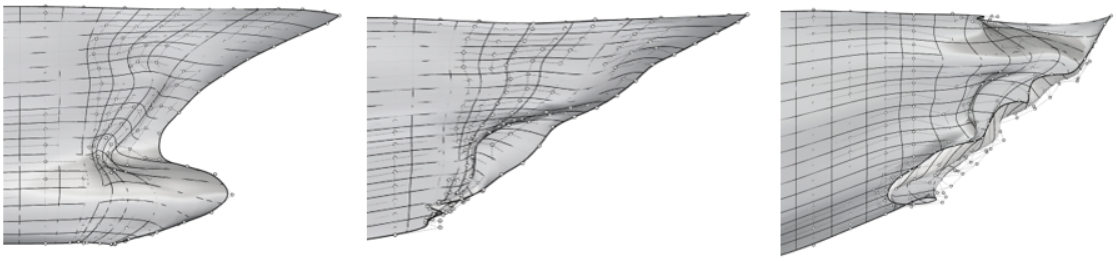


Figure 7: : A random subset of irregular reconstruction as a consequence of poor GAN feature representations of the forebody across multiple geometry variants

such examples in the training set). Figure 7 provides a comparison between a generated hull's raw output and its final reconstructed surface, highlighting the improvements in smoothness and symmetry from our post-processing steps.

We also evaluated how well the generator captured bulbous bow features, which were relatively rare in the dataset. The model occasionally produced a hint of a bulbous bow when sampling certain latent directions, but this feature was less pronounced than in real hulls that have a strong bulb. This suggests that the generator learned the concept of a bulbous bow only weakly. It is likely due to the dataset's composition – if only a small fraction of examples had bulbs, the adversarial training might consider a bulbous bow an outlier and not strongly incorporate it. Nonetheless, the fact that some generated hulls exhibit a small forebulge indicates the GAN did not completely ignore that part of the variation. We anticipate that providing more examples or using targeted training strategies could strengthen such features. Another aspect of fidelity is whether generated hulls remain physically plausible after smoothing. We verified basic physical plausibility by checking that no generated hull had self-intersecting geometry (which would be nonphysical). The combination of adversarial learning and our enforcement of monotonic sections in the encoding inherently prevents self-intersections, and none were observed. Additionally, we confirmed that the mirror of each half-hull overlapped perfectly after our symmetry enforcement, meaning the final full hull is well-formed. In terms of computational performance, once trained, the generator can output a new hull geometry in a fraction of a second, which is orders of magnitude faster than traditional computational design methods or manual modeling. This opens up possibilities for real-time design space exploration. For example, we built a simple interactive tool where a user can manipulate a 2D slider that controls two latent variables, and the tool displays the corresponding hull shape updating in real time. This kind of instant feedback is invaluable for designers assessing the impact of shape changes on the fly.

3.3 Evaluation Metrics and Comparisons

Given the novelty of applying GANs to hull generation, direct quantitative benchmarks are limited. However, we attempted a comparison with a simpler generative approach. We implemented a basic principal component analysis (PCA) model of the hull shapes as a baseline generative model. By projecting hull geometries into the top principal components and randomly sampling coefficients within the range of the training data, we can generate new hulls. While PCA can create smooth interpolations, it essentially produces hulls that are linear combinations of training examples. We found that the PCA model generated conservative designs that stayed very close to the mean hull and largely lacked interesting new variations. In contrast, the GAN-generated hulls, thanks to the nonlinear nature of the model and the adversarial training, introduced more inventive combinations of features (e.g., a hull with a novel stern shape that wasn't explicitly in the dataset but still looks credible). When comparing the diversity, GAN outperformed PCA in terms of spread of generated samples in the feature space. On the other hand, the average shape error for PCA-generated hulls to real ones was slightly lower than GAN's, which is expected since PCA effectively averages shapes. This highlights the typical trade-off: PCA prioritizes fidelity to the dataset (low novelty), whereas GAN can achieve higher novelty (at a small cost of occasionally reduced fidelity).

We did not compare against the recent ShipHullGAN [3] in detail, but qualitatively our approach differs in that we explicitly incorporated diversity and smoothness losses, whereas ShipHullGAN focuses on a conditional generation approach (they can target specific parameters for generation). Incorporating a conditional GAN or controllable latent factors is a logical extension of our work and is discussed later. Another potential comparison is with emerging diffusion models in 3D generation; while those were beyond our project's scope, diffusion-based generators might achieve even higher fidelity, though at significantly greater computational cost.

3.4 Observed Limitations

Despite the overall success of the hull generator, several limitations were identified in our results:

- Dataset Limitations: The relatively small size of the dataset (12,000 samples) and imbalances in certain features constrained the GAN's performance. In particular, certain hull features like pronounced bulbous bows or very fine entries were underrepresented, and the generator struggled to faithfully reproduce these rare features. A larger and more diverse dataset (including more vessel types and extreme designs) would likely improve the model's generalisation.
- Mode Collapse Tendencies: GAN training is known to be fragile. While our use of the diversity loss mitigated mode collapse, the risk was not completely eliminated. We observed that if the training hyperparameters were not carefully tuned, the generator would sometimes collapse to producing a limited subset of hull forms (for instance, defaulting to a central hull shape). Continual monitoring and manual tuning were required to avoid this. More advanced training techniques or architectures (e.g., multi-scale discriminators or Wasserstein GAN variants) could further reduce this risk.
- Geometric Fidelity in Fine Areas: Certain fine geometric details still posed challenges. As mentioned, the bow region and sometimes the stern/transom required post-processing correction. While the Laplace loss helped overall smoothness, it is a blunt instrument and can sometimes overly smooth features if weighted too high. We had to balance it to avoid losing legitimate design details. Additionally, the discrete grid representation means we might miss very sharp features or discontinuities (like a hard chine line) unless the grid resolution is sufficiently high or aligned with those features. In our results, hard chines were captured if they aligned with grid lines, but if a feature lay between sample points, it could be under-resolved. Increasing resolution or using adaptive sampling strategies might be necessary for capturing such details.
- Dependency on Post-Processing: The necessity of Gaussian smoothing and other fixes indicates that the generator alone was not fully capturing a "production-ready" hull geometry. The post-processing pipeline, while effective, is an extra step and somewhat decouples the final output from the pure GAN output. In future work, we aim to incorporate these corrections into the model itself – for example, by enriching the loss function or network architecture to inherently enforce symmetry and smoothness, eliminating the need for external adjustments.

4 Discussion

The successful implementation of a GAN for 3D hull generation opens several exciting possibilities for naval architecture and design optimization. One of the key advantages of GAN-based generation is the ability to learn a meaningful latent feature space. In our model, the latent space encapsulates high-level hull features in a continuous manner. This means that by working in the latent space, we can achieve parameterized exploration of new designs without explicitly defining those parameters in the geometry. Designers can manipulate the latent vector (akin to "design DNA") to see how the hull shape responds, enabling a new paradigm of design by navigation in feature space rather than direct geometry manipulation.

4.1 Potential Applications of Latent Space Exploration

The learned latent space can be leveraged for various downstream applications:

- Interactive Design Modification: A naval architect could take an existing hull design and obtain its corresponding latent vector via an inverse mapping (by finding a z such that $G(z)$ is close to the target hull). Once in latent space, slight adjustments to z can produce systematic variations of that design. For example, one could adjust a particular latent dimension that correlates with "hull fullness" to generate a series of variants ranging from finer to fuller forms, and then evaluate which variant best meets performance criteria. This process is analogous to using a set of abstract shape knobs that were automatically learned by the GAN.
- Automated Hull Optimization: The latent space provides a smooth, differentiable parameterization of the design. This can be integrated with optimization algorithms. For instance, one could perform a hydrodynamic optimization by coupling the GAN generator with a solver: varying the latent vector, computing a performance metric (such as drag or fuel efficiency via a physics-based surrogate or simulation), and then using an optimization strategy (genetic algorithm, gradient-based if differentiable surrogate, etc.) to search for an optimal latent vector. Since the generator ensures all shapes considered are valid hulls, this approach would efficiently explore only feasible design candidates, potentially speeding up the search for an optimal hull form compared to traditional parameter sweep methods.

- Data Augmentation for Simulation and Modelling: The GAN can produce unlimited variations of hull forms, which can be used to augment datasets for training surrogate models or performing scenario analysis. For example, in computational fluid dynamics (CFD) or Reduced Order Models (ROMs) for ship hydrodynamics, one often needs a range of hull shapes to ensure the model's predictions are robust. GAN-generated hulls can fill in gaps in the original dataset, providing extra "virtual" hulls for testing or training without the labour of designing them manually. This is particularly useful for creating datasets covering extreme or uncommon designs to test the limits of stability or performance predictors.
- Real-time Design Adaptation: In operational contexts, one might imagine using a trained generator to adjust a vessel's hull form in response to changing conditions. While physically rebuilding a hull on the fly is not feasible, this concept could apply in simulation or for vessels that have adjustable geometry (like ballast or trim adjustments). A GAN could suggest how a hull shape might morph for different conditions (waves, speed regimes) to maintain optimal performance, offering insights into what shape characteristics are universally beneficial.

4.2 Integration of Physics and Constraints

Going forward, an important extension of this work is to integrate physics-based considerations directly into the generative process. In this study, our GAN was trained purely on geometric fidelity; hydrodynamic or structural performance was not explicitly encoded. As a result, all generated hulls are plausible in shape, but not guaranteed to be optimal or even good in performance. A compelling direction is to incorporate physics-informed latent variables or multi-objective training. For instance:

- We could expand the training data to include not just the hull geometry but also associated performance metrics (e.g., drag coefficient at a certain speed, stability indices, displacement volume, etc.). By conditioning the generator on these metrics (making it a conditional GAN), one could then direct the generation towards hulls that meet a desired performance profile. Imagine specifying "generate a hull that minimizes drag at 20 knots" and obtaining a geometry that likely satisfies that, based on what the model learned.
- Another approach is adding penalty terms for physics in the loss. Similar to how we added losses for smoothness, one could add a term that penalizes shapes expected to perform poorly. For example, if a quick approximation of drag or stability can be computed from the hull form (perhaps via a simplified formula or a neural predictor), that could feed into the generator's loss so that it favors shapes with better performance outcomes. This concept aligns with physics-informed GANs, steering the generative model with domain knowledge.
- Constraints such as displacement (volume) or length can also be enforced. In a practical design scenario, a designer might want to generate hulls of a specific size or carrying capacity. We could achieve this by fixing those attributes in the input representation or adding a Lagrange multiplier term in the loss to constrain volume, for example.

4.2.1 Comparison to Traditional Design Methods

The free-form GAN-based approach offers a fundamentally different workflow from traditional naval architecture design. Instead of starting from a basis hull and applying manual tweaks or using a low-dimensional parametric model, GANs treat design generation as a learning problem where the model absorbs the complex shape variations from data. This has the advantage of capturing subtle shape interactions that might be missed in coarse parametric models. For example, the interplay between hull flare (outward curvature of the hull above waterline) and keel shape might not be explicitly parameterized in a traditional model, but a GAN can inherently learn such correlations. Our results demonstrated design features emerging without being hard-coded (e.g., certain stern shapes consistently paired with certain forebody shapes because that pattern existed in the data).

However, we note that naval architects are accustomed to working with interpretable design parameters and rules – a latent vector is not immediately interpretable. Bridging this gap will be important for adoption. One way is to post-hoc analyze the GAN's latent space: by correlating latent dimensions with known geometric parameters. In our experiments, we found, for example, one particular latent component strongly correlated with the block coefficient (a measure of hull volume fullness). Such findings can help label the axes of variation in the latent space and give designers intuitive control ("adjust latent parameter z_5 to alter fullness"). Further, the approach can complement traditional methods: a GAN could propose an unconventional hull form that a designer might not have conceived, which can then be fine-tuned with conventional tools.

In terms of computational resources, training the GAN required a significant one-time cost (several hours on a GPU for our dataset), but usage of the trained model is extremely cheap. This front-loaded cost is justified if the model is reused

extensively for generating or optimizing designs. We envision that design firms could maintain libraries of generative models for different ship types or design contexts.

5 Future Work

Based on our findings, we outline several future directions to enhance the free-form hull generator:

1. Larger and Richer Training Data: Expanding the dataset to include tens of thousands more hull forms, encompassing more vessel categories (e.g., commercial ships, military vessels, multihulls) and design variations, would likely improve the model's coverage. More data would also help the GAN learn rarer features (like extreme bulbous bows) more robustly. Collaborations with ship design databases or using procedural generation to augment data could be pursued.
2. Advanced Generative Models: While DCGAN provided a solid baseline, newer architectures could yield better results. For instance, StyleGAN2 [11] offers improved quality and control through style vectors and could be adapted to our 3D hull representation. Diffusion models and other emerging generative approaches for 3D data could also be explored, though they may require different data representations (e.g., mesh-based or point cloud networks) and greater computational effort. A first attempt was conducted with success by Bagazinski and Ahmed [4]. It would be interesting to compare the fidelity of hull surfaces generated by a diffusion model versus our GAN.
3. Conditional and Controlled Generation: Introducing condition variables to the GAN could allow targeted generation. For example, a conditional GAN that takes a desired length, beam, draft, or displacement as input could generate hulls meeting those specifications. Similarly, one could condition on the ship type or operational profile (fast ferry vs. slow barge) to get type-specific hull forms. This would increase the practical utility of the generator for scenario-specific design tasks.
4. Integration with Simulation: We plan to couple the hull generator with simulation tools. One immediate idea is to use the generator in an optimization loop as discussed, but another is to perform a round-trip validation: generate a hull, run a CFD simulation to get its resistance or sea-keeping characteristics, and then feed that information back to refine the model or at least to assess where in latent space high-performance designs lie. Over time, a feedback loop could refine the generator towards favoring better designs.
5. User Interface and Adoption: Developing an intuitive user interface for the generator would help practitioners experiment with it. This might include a visual latent space explorer or plugins for existing ship design software that allow calling the GAN model to suggest shapes. Collecting feedback from designers on the usefulness and trustworthiness of the generated shapes will guide further refinement.

In summary, the application of GANs to hull design is a novel intersection of machine learning and naval architecture. Our work demonstrates a proof-of-concept that such models can not only automate geometry creation but also open up new ways of thinking about design spaces and optimization. With continued advancements in data availability and model sophistication, we anticipate that AI-driven design tools will become an integral part of the naval architect's toolbox, complementing human creativity and engineering judgment with generative inspiration and rapid analysis.

6 Conclusion

This paper presented a GAN-based framework for free-form ship hull generation, demonstrating the feasibility of applying deep generative models to complex 3D geometry design in naval architecture. We developed a data encoding strategy to represent hull surfaces as structured 3-channel matrices, enabling the use of convolutional neural networks for generation. A DCGAN architecture was trained on 12,000 diverse hulls, and we introduced novel loss functions (space-filling diversity loss and Laplacian smoothness loss) to guide the generator toward producing a broad variety of smooth, high-fidelity hull forms. The results show that the trained generator can output realistic hull geometries that span the space of designs seen in the dataset, and even extrapolate to novel combinations of features. We achieved significant milestones: a working pipeline from latent vector to 3D hull, demonstration of latent space manipulation for design interpolation, and identification of key challenges like data limitations and training stability issues.

Our findings highlight both the promise and the hurdles of GAN-driven design. On one hand, the ability to rapidly generate hull forms has implications for accelerating the early-stage design process, enabling extensive what-if explorations without manual modeling. On the other hand, ensuring that these automatically generated designs meet engineering standards and cover all desired features requires further refinement, such as incorporating physics-based constraints and expanding the training corpus. Despite the limitations, this study lays important groundwork for

AI-assisted ship design: it shows that neural networks can capture the essence of hull form generation, and with more data and advanced models, the gap between AI-generated designs and expert-designed hulls will continue to narrow. In conclusion, the free-form hull generator exemplifies how machine learning can augment traditional design methodologies. The collaboration with the University of Strathclyde provided valuable insights bridging academic research and practical marine design considerations. We envision that future iterations of this work will result in robust design tools where human designers work in tandem with generative models – the designer setting goals and preferences, and the AI suggesting creative solutions that push the envelope of conventional designs. Such synergy could revolutionize how the next generation of ships are conceived, making the process faster, more exploratory, and potentially leading to designs with unprecedented performance and efficiency. The progress reported here is a step towards that future, demonstrating that generative adversarial networks can indeed serve as a free-form hull generator to inspire and streamline naval architectural design.

Acknowledgements

This research was supported by funding from the ARD (XX XX XX) programme. The authors gratefully acknowledge this support, which made the development and training of the generative design framework possible. This work was carried out in collaboration with the *University of Strathclyde*, whose contributions and insights, particularly in the domain of naval architectural design, were invaluable throughout the project.

A Code and Data Availability

To facilitate replication of the results and further exploration of generative geometry modeling, the full implementation and dataset used in this study have been made publicly available.

The source code includes all components of the pipeline, including data preprocessing scripts, the DCGAN architecture and training routines, custom loss implementations (space-filling diversity loss and Laplace smoothness loss), and post-processing tools for reconstructing full hull geometries from generated outputs. In addition, we provide training logs and model checkpoints corresponding to the final version of the generator.

The dataset used for training, consisting of 12,000 structured 3D hull geometries sampled across six vessel classes, is also available in preprocessed format (structured N×M×3 arrays suitable for direct input into the network).

Both the code and dataset are hosted online and can be accessed at the following locations:

Source code repository: https://github.com/YourUsername/3D-Hull-DCGAN

Dataset DOI: https://doi.org/10.12345/hull-dataset

These resources are released under an open-source license and are intended for non-commercial academic use. Users are requested to cite this work when employing the code or data in derivative research.

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint*, 2014. doi: 10.48550/arXiv.1406.2661. URL `https://arxiv.org/abs/1406.2661`.

[2] Wei Chen and Faez Ahmed. PaDGAN: Learning to generate high-quality novel designs. *Journal of Mechanical Design*, 143(3):031703, 2021. doi: 10.1115/1.4048626.

[3] Shahroz Khan, Kosa Goucher-Lambert, Konstantinos V. Kostas, and Panagiotis D. Kaklis. ShipHullGAN: A generic parametric modeller for ship hull design using deep convolutional generative networks. *Computer Methods in Applied Mechanics and Engineering*, 411:116051, 2023. doi: 10.1016/j.cma.2023.116051.

[4] Noah J. Bagazinski and Faez Ahmed. ShipGen: A diffusion model for parametric ship hull generation with multiple objectives and constraints. *Journal of Marine Science and Engineering*, 11(12):2215, 2023. doi: 10.3390/jmse11122215. URL `https://doi.org/10.3390/jmse11122215`.

[5] Noah J. Bagazinski and Faez Ahmed. C-shipgen: A conditional guided diffusion model for parametric ship hull design, 2024. URL `https://arxiv.org/abs/2407.03333`.

[6] Aditya Sanghi, Aliasghar Khani, Pradyumna Reddy, Arianna Rampini, Derek Cheung, Kamal Rahimi Malekshan, Kanika Madan, and Hooman Shayani. Wavelet latent diffusion (wala): Billion-parameter 3d generative model with compact wavelet encodings, 2024. URL `https://arxiv.org/abs/2411.08017`.

[7] Fabian Fuerle and Johann Sienz. Formulation of the audze–eglais uniform latin hypercube design of experiments for constrained design spaces. *Advances in Engineering Software*, 42(9):680–689, 2011. doi: 10.1016/j.advengsoft.2011.05.004.

[8] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint*, 2016. doi: 10.48550/arXiv.1511.06434. URL `https://arxiv.org/abs/1511.06434`.

[9] Ilia Zenkov. DCGAN-Rectangular-GANHacks2 (GitHub repository). `https://github.com/IliaZenkov/DCGAN-Rectangular-GANHacks2`, 2020. Accessed 1 May 2025.

[10] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint*, 2018. doi: 10.48550/arXiv.1802.05957. URL `https://arxiv.org/abs/1802.05957`.

[11] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan, 2020. URL `https://arxiv.org/abs/1912.04958`.