A. 3D Transformations and Their Relaxations

In this section, we define the semantic transformations that we consider (App. A.1) and provide the taylor relaxations for the transformations (App. A.2), as well as their Jacobians for DeepG3D (App. A.4). We also give some background on the interval arithmetic used to compute bounds on the approximation error (App. A.3).

A.1. Semantic Transformations

3DCertify can handle a wide range of semantic transformations, including 3D rotation around any axis with $\theta \in \mathbb{R}$ as defined in Eq. (5). We can also certify shearing, twisting, and tapering of a point cloud, defined pointwise (since each point is transformed independently) for a point $\boldsymbol{p} = (x, y, z)^T$ as

$$\text{Shear}(\boldsymbol{p}, \boldsymbol{\theta}) = \begin{pmatrix} \theta_1 z + x \\ \theta_2 z + y \\ z \end{pmatrix}$$

$$\text{Twist}(\boldsymbol{p}, \theta) = \begin{pmatrix} x\cos(\theta z) - y\sin(\theta z) \\ x\sin(\theta z) + y\cos(\theta z) \\ z \end{pmatrix}$$

$$\text{Taper}(\boldsymbol{p}, \boldsymbol{\theta}) = \begin{pmatrix} (\frac{1}{2}\theta_1^2 z + \theta_2 z + 1)x \\ (\frac{1}{2}\theta_1^2 z + \theta_2 z + 1)y \\ z \end{pmatrix}$$

or any composition of these functions.

A.2. Taylor Relaxations

In Section 3.2, we presented the general form of our linear bounds $f_l(P, \boldsymbol{\theta})$ and $f_u(P, \boldsymbol{\theta})$ for any twice continuously differentiable transformation function $f(P, \boldsymbol{\theta})$, as well as the relaxation for Rot_Z as an example. Rotation around the other two axes, *i.e.*, Rot_X and Rot_Y can be computed analogously. Here, we list the linear relaxations for the remaining transformation functions we use in our experiments.

All transformations can be applied to each point individually, allowing us to denote them for a single point as $f(\boldsymbol{p}, \boldsymbol{\theta})$ with $\boldsymbol{p} = (x, y, z)^T \in P$. For each transformation, we list the first-order taylor polynomial $Q(\boldsymbol{p}, \boldsymbol{\theta})$ and remainder $R(\boldsymbol{p}, \boldsymbol{\theta})$, such that $f(\boldsymbol{p}, \boldsymbol{\theta}) = Q(\boldsymbol{p}, \boldsymbol{\theta}) + R(\boldsymbol{p}, \boldsymbol{\theta})$. As described in Section 3.2, we use interval arithmetic (App. A.3) to get real-valued bounds $\boldsymbol{l}_R \leq R(\boldsymbol{p}, \bar{\boldsymbol{\theta}}) \leq \boldsymbol{u}_R$ for the interval $\bar{\boldsymbol{\theta}} = [\boldsymbol{l}_{\boldsymbol{\theta}}, \boldsymbol{u}_{\boldsymbol{\theta}}]$ with $\boldsymbol{t} = (\boldsymbol{l}_{\boldsymbol{\theta}} + \boldsymbol{u}_{\boldsymbol{\theta}})/2$ and therefore the lower constraint $f_l(\boldsymbol{p}, \boldsymbol{\theta}) = Q(\boldsymbol{p}, \boldsymbol{\theta}) + \boldsymbol{l}_R$ and upper constraint $f_u(\boldsymbol{p}, \boldsymbol{\theta}) = Q(\boldsymbol{p}, \boldsymbol{\theta}) + \boldsymbol{u}_R$.

Shearing:

$$\begin{aligned} Q_{\text{Shear}}(\boldsymbol{p}, \boldsymbol{\theta}) &= \begin{pmatrix} t_1 z + x \\ t_2 z + y \\ z \end{pmatrix} \\ &\quad + \begin{pmatrix} z \\ 0 \\ 0 \end{pmatrix} (\theta_1 - t_1) + \begin{pmatrix} 0 \\ z \\ 0 \end{pmatrix} (\theta_2 - t_2) \\ R_{\text{Shear}}(\boldsymbol{p}, \bar{\boldsymbol{\theta}}) &= 0 \end{aligned}$$

Twisting:

$$\begin{aligned} Q_{\text{Twist}}(\boldsymbol{p}, \theta) &= \begin{pmatrix} \cos(tz)x - \sin(tz)y \\ \sin(tz)x + \cos(tz)y \\ z \end{pmatrix} \\ &\quad + \begin{pmatrix} -z(\sin(tz)x + \cos(tz)y) \\ z(\cos(\theta z)x - \sin(\theta z)y) \\ 0 \end{pmatrix} (\theta - t) \\ R_{\text{Twist}}(\boldsymbol{p}, \bar{\theta}) &= \frac{1}{2} \begin{pmatrix} -z^2(\cos(\bar{\theta} z)x - \sin(\bar{\theta} z)y) \\ -z^2(\sin(\bar{\theta} z)x + \cos(\bar{\theta} z)y) \\ 0 \end{pmatrix} (\bar{\theta} - t)^2 \end{aligned}$$

Tapering:

$$\begin{aligned} Q_{\text{Taper}}(\boldsymbol{p}, \boldsymbol{\theta}) &= \begin{pmatrix} (\frac{1}{2}t_1^2 z + t_2 z + 1)x \\ (\frac{1}{2}t_1^2 z + t_2 z + 1)y \\ z \end{pmatrix} \\ &\quad + \begin{pmatrix} t_1 zx \\ t_1 zy \\ 0 \end{pmatrix} (\theta_1 - t_1) + \begin{pmatrix} zx \\ zy \\ 0 \end{pmatrix} (\theta_2 - t_2) \\ R_{\text{Taper}}(\boldsymbol{p}, \bar{\boldsymbol{\theta}}) &= \frac{1}{2} \begin{pmatrix} zx \\ zy \\ 0 \end{pmatrix} (\bar{\theta_1} - t_1)^2 \end{aligned}$$

A.3. Interval Arithmetic

When evaluating functions such as $R(P, \boldsymbol{\theta})$ on intervals, we use the following standard operators:

$$\begin{aligned} -[x_l, x_u] &= [-x_u, -x_l] \\ [x_l, x_u] + [y_l, y_u] &= [x_l + y_l, x_u + y_u] \\ [x_l, x_u] - [y_l, y_u] &= [x_l - y_u, x_u - y_l] \\ [x_l, x_u] \cdot [y_l, y_u] &= [\min(x_l y_l, x_l y_u, x_u y_l, x_u y_u), \\ &\qquad \max(x_l y_l, x_l y_u, x_u y_l, x_u y_u)] \end{aligned}$$

For mixed operations with scalars, *i.e.*, $a * [x_l, x_u]$, we can treat the scalar as an interval with one element: $[a, a] * [x_l, x_u]$.

In addition to these basic operators, we use the following

sine function:

$$\sin([x_l, x_u]) = [y_l, y_u], \text{ where}$$

$$y_l = \begin{cases} -1 & -\frac{\pi}{2} + 2k\pi \in [x_l, x_u] \\ \min(\sin(x_l), \sin(x_u)) & \text{otherwise} \end{cases}$$

$$y_u = \begin{cases} 1 & \frac{\pi}{2} + 2k\pi \in [x_l, x_u] \\ \max(\sin(x_l), \sin(x_u)) & \text{otherwise} \end{cases}$$

with $k \in \mathbb{Z}$. Similarly, we can define the cosine function:

$$\cos([x_l, x_u]) = [y_l, y_u], \text{ where}$$

$$y_l = \begin{cases} -1 & \pi + 2k\pi \in [x_l, x_u] \\ \min(\cos(x_l), \cos(x_u)) & \text{otherwise} \end{cases}$$

$$y_u = \begin{cases} 1 & 2k\pi \in [x_l, x_u] \\ \max(\cos(x_l), \cos(x_u)) & \text{otherwise.} \end{cases}$$

To compute the square x^2 of $x = [x_l, x_u]$, we could simply use $x \cdot x$. While sound, we can compute a tighter interval for some cases:

$$[x_l, x_u]^2 = \begin{cases} [x_l^2, x_u^2] & x_l \geq 0 \\ [x_u^2, x_l^2] & x_u \leq 0 \\ [0, \max(x_l^2, x_u^2)] & \text{otherwise.} \end{cases}$$

Using these operators and functions, we can evaluate all of our relaxations and their derivatives with intervals as input.

A.4. Jacobian Matrices

Computing linear relaxations using Deepg3D (Section 3.1) requires the Jacobians of our 3D transformations, both with respect to the transformation parameters and with respect to the point cloud inputs. For example, for 3D rotation around the z-axis with $\theta \in \mathbb{R}$, as defined in Eq. (5), we compute

$$\partial_{\boldsymbol{p}} \mathrm{Rot}_Z(\boldsymbol{p}, \theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{10}$$

and

$$\partial_{\theta} \mathrm{Rot}_Z(\boldsymbol{p}, \theta) = \begin{pmatrix} -x\sin(\theta) - y\cos(\theta) \\ x\cos(\theta) - y\sin(\theta) \\ z \end{pmatrix}. \tag{11}$$

The corresponding Jacobians for Shear, Twist and Taper (Section 4.1) are given by:

$$\partial_{\boldsymbol{p}} \mathrm{Shear}(\boldsymbol{p}, \boldsymbol{\theta}) = \begin{pmatrix} 1 & 0 & \theta_1 \\ 0 & 1 & \theta_2 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\partial_{\boldsymbol{\theta}} \mathrm{Shear}(\boldsymbol{p}, \boldsymbol{\theta}) = \begin{pmatrix} z & 0 \\ 0 & z \\ 0 & 0 \end{pmatrix}$$

$$\partial_{\boldsymbol{p}} \mathrm{Twist}(\boldsymbol{p}, \theta) = \begin{pmatrix} \cos(\theta z) & -sin(\theta z) & -\theta(\sin(\theta z)x + \cos(\theta z)y) \\ \sin(\theta z) & \cos(\theta z) & \theta(\cos(thetaz)x - \sin(\theta z)y) \\ 0 & 0 & 1 \end{pmatrix}$$

$$\partial_{\theta} \mathrm{Twist}(\boldsymbol{p}, \theta) = \begin{pmatrix} -z(\sin(\theta z)x + \cos(\theta z)y) \\ z(\cos(\theta z)x - \sin(\theta z)y) \\ 0 \end{pmatrix}$$

$$\partial_{\boldsymbol{p}} \mathrm{Taper}(\boldsymbol{p}, \boldsymbol{\theta}) = \begin{pmatrix} \frac{1}{2}\theta_1^2 z + \theta_2 z + 1 & 0 & (\frac{1}{2}\theta_1^2 + \theta_2)x \\ 0 & \frac{1}{2}\theta_1^2 z + \theta_2 z & (\frac{1}{2}\theta_1^2 + \theta_2)y \\ 0 & 0 & 1 \end{pmatrix}$$

$$\partial_{\boldsymbol{\theta}} \mathrm{Taper}(\boldsymbol{p}, \boldsymbol{\theta}) = \begin{pmatrix} \theta_1 zx & zx \\ \theta_1 zy & zy \\ 0 & 0 \end{pmatrix}$$

B. Proof for Composition of Transformations

To show that our taylor approximations introduced in Section 3.2 can be applied to the composition of multiple transformations, we show that the composition of two twice continuously differentiable functions is itself twice continuously differentiable. That is, given two twice continuously differentiable functions $f : \mathbb{R}^n \mapsto \mathbb{R}^p$ and $g : \mathbb{R}^m \mapsto \mathbb{R}^n$, we want to show that $h = f \circ g$ is also twice continuously differentiable.

To simplify notation, we define $\boldsymbol{y} = f(\boldsymbol{u})$ and $\boldsymbol{u} = g(\boldsymbol{x})$. Using the chain rule, we know that the first-order derivatives exist and can, with slight liberties in notation, be written as:

$$\frac{\partial \boldsymbol{y}}{\partial x_i} = \sum_k \frac{\partial \boldsymbol{y}}{\partial u_k} \frac{\partial u_k}{\partial x_i}. \tag{12}$$

Furthermore, we know that f and g are twice differentiable, hence Eq. (12) consists of compositions, products and sums of differentiable functions and thus is again differentiable. We therefore conclude that $f \circ g$ is itself twice differentiable.

It remains to be shown that the second-order derivatives are continuous. Using Faà di Bruno's formula, we can write the second-order derivatives as:

$$\frac{\partial^2 \boldsymbol{y}}{\partial x_i \partial x_j} = \sum_k \frac{\partial y}{\partial u_k} \frac{\partial^2 u_k}{\partial x_i \partial x_j} + \sum_k \sum_l \frac{\partial^2 \boldsymbol{y}}{\partial u_k \partial u_l} \frac{\partial u_k}{\partial x_i} \frac{\partial u_l}{\partial x_j}. \tag{13}$$

We know that f, g and their first- and second-order derivatives are continuous. Equation (13) is therefore a combination of compositions, products and sums of continuous functions, which means it is itself continuous. This means $f \circ g$ is twice continuously differentiable and we can therefore calculate Taylor bounds for any composition of twice continuously differentiable transformations.

C. PointNet Architectures

For both object classification and part segmentation, we use PointNet [36] models. Below we present the exact layer configurations used.

Object Classification

For object classification, we use the following network architecture:

No	Type	Normalization	Activation	Features
1	Linear	BatchNorm	ReLU	64
2	Linear	BatchNorm	ReLU	64
3	Linear	BatchNorm	ReLU	64
4	Linear	BatchNorm	ReLU	128
5	Linear	BatchNorm	ReLU	1024
6	MaxPool			1024
7	Linear	BatchNorm	ReLU	512
8	Linear	BatchNorm	ReLU	256
9	Linear		SoftMax	num classes

The first block of linear (fully connected) layers (no 1 to 5) is executed on each point individually, but sharing weights across all points. We implement this via 1D convolution layers with stride 1 as in the original work by Qi *et al*. [36]. Layer 6 pools each feature across points. During training, a dropout of 30% is applied for layer 8.

Part Segmentation

No	Type	Normalization	Activation	Features
1	Linear	BatchNorm	ReLU	64
2	Linear	BatchNorm	ReLU	128
3	Linear	BatchNorm	ReLU	256
4	Linear	BatchNorm		128
5	MaxPool			128
6	Repeat			128
7	Concatenate (1, 2, 3, 6)			576
8	Linear	BatchNorm	ReLU	256
9	Linear	BatchNorm	ReLU	128
10	Linear		SoftMax	num parts

The architecture for part segmentation differs in some ways, since it needs to predict a label for each point individually. Again, the first block of linear layers (1-4) is applied to each point individually with shared weights and max pool combines per-point features to one global feature vector. Layer 7 concatenates the local features of layers 1 to 3 with the global feature from layer 5 for each point by simple concatenation. The last 3 linear layers are again applied individually for each point on the combined local and global feature and predict the part the particular point belongs to.

Since DeepPoly [44] cannot handle this architecture for part segmentation, we implement novel relaxations for the concatenation and repeat layers. In particular, the transformer for concatenation requires the verifier to handle layers with multiple predecessors, which is out of reach for current state-of-the-art verifiers. We provide our implementation in the accompanying code.

ϵ	0.005	0.010	0.015
DeepPoly [44]	72.8	33.3	3.7
Ours	**79.0**	**38.3**	**6.2**

Table 6. Percentage of certified images with different max pool relaxations for two different image classification tasks for ℓ_∞ noise perturbations.

Group Size	Certified (%)	Time (s)
4	93.5	56
8	93.5	76
12	93.5	119

Table 7. Percentage of certified point clouds with 64 points for different max pool group sizes for $\pm 3°$ rotation.

D. Additional Experiments

In this section, we present additional empirical evidence for the benefits of our improved max pool relaxations in App. D.1, and investigate the effect different max pool group sizes have on certification results. We also show that Taylor3D efficiently scales to real-world point cloud sizes in App. D.2, with running times of only a few milliseconds.

D.1. Improved Max Pool Relaxation

Applications beyond point clouds In Section 4.2, we show that our improved max pool relaxation, introduced in Section 3.3, significantly improves certification for PointNet models compared to the previous state-of-the-art, especially for models with larger pooling layers. Here, we demonstrate that our new relaxations are useful beyond PointNet, *i.e*., for any network architecture containing max pool layers. To that end, we show certification results for a convolutional image classification model with nine conv/-linear layers and two max pool layers for the MNIST [21] dataset in Table 6, comparing our improved relaxations with the best baseline. Our improved max pool relaxations consistently outperform the previous state-of-the-art across all ϵ-values, significantly increasing the number of images for which we can certify correct classification. These results demonstrate that models beyond the 3D point cloud domain benefit from our new relaxations.

Max pool group size Our improved max pool relaxation, introduced in Section 3.3, requires computing the convex hull of the polyhedral relaxation, for which the running time grows exponentially in the number of input neurons. This

	Rot_Z		Twist	
Points	Taylor3D	DeepG3D	Taylor3D	DeepG3D
100 000	0.036	393	0.070	366
200 000	0.070	887	0.169	848
300 000	0.104	1502	0.266	1496

Table 8. Running time in seconds to compute the relaxations for different real-world point cloud sizes with Taylor3D and DeepG3D. Taylor3D achieves speed-ups of up to 14 442x for Rot_Z and 5 624x for Twist.

is why we recursively split the max pool operation into sub groups. Table 7 shows the certification accuracy and average running time of DeepPoly with the improved max pool relaxation for different group sizes. Increasing the group size beyond this range is impractical (*i.e.*, more than 3h per point clouds) due to the exponential scaling behavior of convex hull computation. Nevertheless, our experiments indicate that our recursively partitioned relaxation is not impeded by this constraint since the different group sizes do not influence certification performance (while, in theory, computing the relaxation over all inputs should be most precise), allowing us to optimize for improved running time.

D.2. Scaling

3D processing of LIDAR point cloud data is an active area of research. The main challenge is the huge size of point clouds (in the order of 100k points [15]) that have to be processed in real-time for most applications. Both DeepG3D and Taylor3D scale linearly with point cloud size and can be parallelized perfectly across points. Table 8 shows the running time of computing linear relaxations for large point cloud sizes using Taylor3D and Deepg3D respectively. All experiments are run on an AMD EPYC 7601 processor with 2.2 GHz. Taylor3D is efficiently implemented as vectorized operations using Numpy [18] and run on a single thread. DeepG3D uses the original parallelized implementation [2] and runs in parallel with 16 threads. The results show that, while both implementations scale linearly in the point cloud size, Taylor3D is significantly more efficient, computing relaxations in only a few milliseconds event for large point cloud sizes on a single thread, thereby achieving speed-ups of up to 14 442x over DeepG3D. This enables easy and efficient scaling to real-world applications.

E. Max Pool Analysis

The state-of-the-art linear relaxations for max pool are imprecise, especially for many inputs to the pooling layer. We demonstrate this by plotting the mean divergence between DeepPoly's upper and lower bounds for each of PointNet's layers in Fig. 2, where we observe that the bounds start to significantly diverge after the max pool layer.

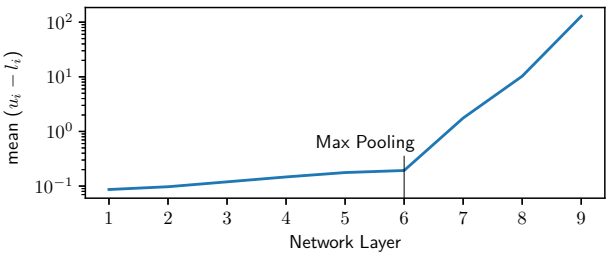

Figure 2. Plotting the mean difference between upper and lower bounds of neurons after each layer shows that the precision significantly decreases after the max pool and therefore motivates the need for improvement. Note the logarithmic scaling of the y-axis.