Text to Graphics by Program Synthesis with Error Correction

Ivan Nikitovic
Boston University
in@bu.edu

Trisha Anil
Boston University
tcanil@bu.edu

Shwondarya Madhavan
Boston University
shmadhav@bu.edu

Arvind Raghavan
Columbia University
ar4284@columbia.edu

Zad Chin
Harvard University
zadchin@college.harvard.edu

Alexander E. Siemenn
MIT
asiemenn@mit.edu

Saisamrit Surbehera
Columbia University
ss6365@columbia.edu

Yann Hicke
Cornell University
ylh8@cornell.edu

Ed Chien
Boston University
edchien@bu.edu

Ori Kerret
Ven Commerce
ori@ven.com

Tonio Buonassisi
MIT
buonassi@mit.edu

Armando Solar-Lezama
MIT
armando@csail.mit.edu

Iddo Drori
MIT, Columbia University, Boston University
idrori@mit.edu,idrori@cs.columbia.edu

Abstract

Using advanced graphics packages requires domain expertise and experience. We demonstrate a text-to-graphics pipeline using GPT models suitable for novice users. Current text-to-image methods fail on visual tasks that require precision or a procedural specification. DALL-E 2 and StableDiffusion cannot accomplish precise design, engineering, or physical simulation tasks. We correctly perform such tasks by turning them into programming tasks, automatically generating code using graphics libraries and running the code to render images and animations. Code generation models often generate errors on complex programs, so we perform local error correction. Rather than subjectively evaluating results on a set of prompts, we generate a new multi-task benchmark of challenge tasks. We demonstrate the applicability of our approach for precise and procedural rendering, animations, and physical simulations using diverse programming languages and graphics environments.

1. Introduction

Text-to-image methods such as DALL-E 2 [20] and StableDiffusion [22] are remarkably effective at producing creative and novel images that reflect a user prompt. The results have captured the popular imagination, and artists are already using these systems as tools in their creative process [21]. However, these methods have various failure modes that disrupt this process. For example, they often fail to produce precise results and respond appropriately to specified numeracy, positions, spellings, etc. They also do not allow for simple compositional changes to the image, such as modifying the color or styling of a particular object without manual marking or outlining. Finally, they may produce images that defy conventional physics.

We address these particular issues by reframing the problem as one of graphics code generation. When a scene is specified at the code level, precision in numerics and positioning is required, objects are instantiated separately and specified, and realistic physics models are generated. We use GPT models [4, 16] to generate Python and JavaScript code for rendering objects in the Browser and Blender [7] and leverage *program synthesis*, allowing for user-guided edits and automatic error correction. Figures 1 and 2 shows a schematic of this.

To demonstrate the efficacy of our method, we show that it outperforms DALL-E 2 and StableDiffusion on tasks from a recent benchmark designed to test the ability of these systems to cope with challenging prompts. In summary, our contributions are listed below:

- We reframe the problem of image generation as one of graphics code generation, allowing for unparalleled precision and compositional control over the image output.
- We leverage program synthesis tools, allowing for automatic error correction and user-guided modification

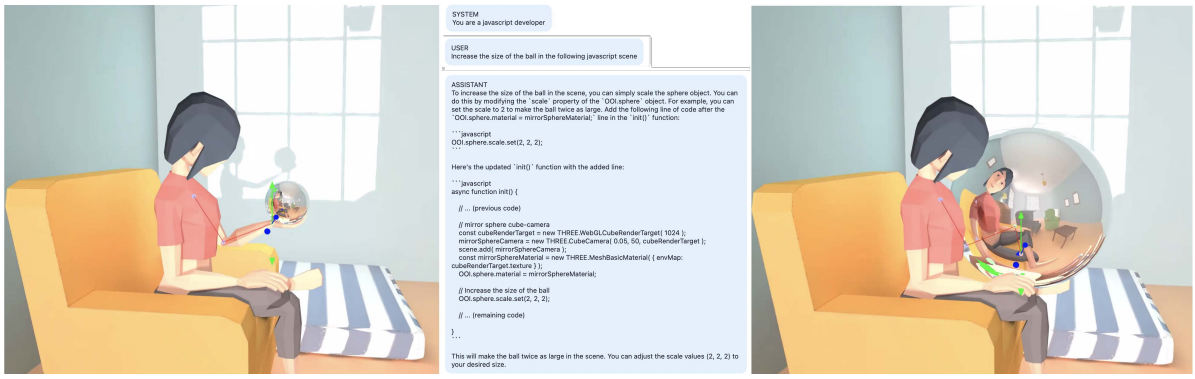

Figure 1. The image on the left is rendered by a JavaScript animation using the three.js library [2]. We pass the corresponding source code to GPT-4, with a prompt to increase the size of the ball, and GPT-4 outputs the modified code. The generated code renders the image on the right which includes self-reflection in a large ball.

of the code (and thus the scene).

- We demonstrate that our method outperforms the existing state-of-the-art on a benchmark of challenging prompts

Lastly, the generative creation of 3D scenes allows for much broader domains of application that operate in the realm of 3D output. For example, we expect it to be useful for architectural, interior, and industrial design applications.

Related Work

There is often thought that humans are generalists, whereas machines are specialists. However, large language models based on transformers such as GPT-3 [3], Gopher [19], and Palm [6], also called foundation models, are generalist learners. Leveraging the concurrent developments in foundation models [3], multimodal learning [18] for bridging vision and language domains, and diffusion models for image generation tasks, text-to-image, or text-conditioned image generation, methods increased their level of photorealism. Text-to-image models may be roughly split into two types: autoregressive transformer-based models [28] and diffusion-based models [24]. Prior state-of-the-art [1, 11, 12, 23] handles specific limitations of text-to-image models such as generating an image within context or modifying object attributes automatically. Text-to-image models are commonly evaluated by the Inception Score (IS) and the Fréchet Inception Distance (FID). Both of these metrics are based on Inception v3 classifier. These measures, therefore, are designed for the unconditional setting and are primarily trained on single-object images. Several approaches rectify these shortcomings. A comprehensive and quantitative multi-task benchmark for text-to-image synthesis does not exist that covers a diverse set of tasks with varying difficulty levels. Imagen [24] introduced DrawBench, a benchmark with 11 categories with approximately 200 prompts total. Human raters (25 participants) were asked to choose a better set of generated images from two models regarding image fidelity and image-text alignment. Categories are: colors counting, conflicting, DALL-E 2, description, misspellings, positional, rare words, Reddit, text. DALL-E 2-Eval [5] proposed PaintSkills to test skills of the generative models — specified object generation, counting, color, and spatial relations. It utilizes the Unity engine to test these tasks using predefined sets of objects, a subset of MS-COCO [15] objects, colors, and spatial relations. Localized Narratives [17] is a multi-modal image captioning approach that can be adapted to benchmarking images. Text captions are first generated by human annotators whose cursor movement and voice commentary hover their cursor over the image to provide richness and accuracy. PartiPrompts, a holistic benchmark of 1,600 English prompts [29], compared to Localized Narratives, is better in probing model capabilities on open-domain text-to-image generation. There are several ways to improve the mathematical reasoning ability of language models: (1) using chain-of-thought (CoT) prompting [13, 27], (2) using the top-k ranking solutions [14] and merging them by voting [26] or least-to-most prompting [30], and (3) using program synthesis and few-shot learning to generate code that answers questions [9]. In this work we use program synthesis and few-shot learning for rendering.

2. Methods

Figure 2 illustrates a schematic of our workflow. To realize ”text to graphics,” two critical components are needed: (1) the ability to correct errors in code that can arise during code generation; (2) the ability to edit code and add to code in response to user prompts. Both features are illustrated in Figure 2) for the case of a blue cube. On the top row, the first instance of generated code may not run properly, requiring error correction and a second rendering attempt.

Step	Output
1. Prompt Inject restart text	`Render a cube in Blender.` `bpy.ops.render.render( write_still =True)`

2. Run Codex in `Complete Mode`	`import bpy`
	`import math`
	`# Delete default cube`
	`bpy.data.objects['Cube'].select = True`
	`bpy.ops.object.delete()`
	`# Add a new cube`
	`bpy.ops.mesh.primitive_cube_add(location=(0, 0, 0))`
	`# Get the object`
	`cube = bpy.context.object`
	`# Set the location`
	`cube.location = (0, 0, 0)`
	`# Set the dimensions`
	`cube.dimensions = (1, 1, 1)`
	`# Set the rotation`
	`cube.rotation_euler = (math.radians(45), math.radians(45), math.radians(45))`
	`# Set the material`
	`mat = bpy.data.materials.new(name="MaterialName")`
	`cube.data.materials.append(mat)`
	`# Render the image`
	`bpy.context.scene.render.filepath = 'cube.png'`
	`bpy.ops.render.render( write_still =True)`

3. Run program, terminates with error	`Error: Python: Traceback (most recent call last):`
	`File "/Users/develop/blender/cube.py", line 5, in <module>`
	`bpy.data.objects['Cube'].select = True`
	`AttributeError: 'Object' object has no attribute 'select'`

4. Fix error by traceback in Codex `Edit Mode`	`Fix [Error from Step 3]`
5. Fixed code diff	`- bpy.data.objects['Cube'].select = True` `+ bpy.data.objects['Cube'].select_set(True)`
6. Run program, correct output rendering	
7. Instructions in Codex `Edit Mode`	`Make the cube blue.`

8. Modified code	`import bpy`
	`import math`
	`# Delete default cube`
	`bpy.data.objects['Cube'].select_set(True)`
	`bpy.ops.object.delete()`
	`# Add a new cube`
	`bpy.ops.mesh.primitive_cube_add(location=(0, 0, 0))`
	`# Get the object`
	`cube = bpy.context.object`
	`# Set the location`
	`cube.location = (0, 0, 0)`
	`# Set the dimensions`
	`cube.dimensions = (1, 1, 1)`
	`# Set the rotation`
	`cube.rotation_euler = (math.radians(45), math.radians(45), math.radians(45))`
	`# Set the material`
	`mat = bpy.data.materials.new(name="MaterialName")`
	`cube.data.materials.append(mat)`
	`# Make the cube blue`
	`mat.diffuse_color = (0.0, 0.0, 1.0)`
	`# Render the image`
	`bpy.context.scene.render.filepath = 'cube.png'`
	`bpy.ops.render.render( write_still =True)`
	`# Set the dimensions`
	`cube.dimensions = (1, 1, 1)`
	`# Set the rotation`
	`cube.rotation_euler = (math.radians(45), math.radians(45), math.radians(45))`
	`# Set the material`
	`mat = bpy.data.materials.new(name="MaterialName")`
	`cube.data.materials.append(mat)`
	`# Render the image`
	`bpy.context.scene.render.filepath = 'cube.png'`
	`bpy.ops.render.render( write_still =True)`

9. Run program, terminates with error	`Error: Python: Traceback (most recent call last):`
	`File "/Users/develop/blender/cube-blue.py", line 32, in <module>`
	`mat.diffuse_color = (0.0, 0.0, 1.0)`
	`ValueError: bpy_struct: item.attr = val: sequences of dimension 0 should contain 4 items, not 3`

10. Fix error using traceback in Codex `Edit Mode`	`Fix [Error from Step 9]`
11. Fixed code diff	`- mat.diffuse_color = (0.0, 0.0, 1.0)` `+ mat.diffuse_color = (0.0, 0.0, 1.0, 1.0)`
12. Run program, correct output rendering	

Table 1. We present an iterative method for image generation, via code generation. The method takes in a user prompt, with an appropriate prefix (like "A Blender rendering of…"), and produces code that generates an image. In the example above, the system generates Python code for a rendering of a cube with Blender. The resultant code may fail in one of two ways: (1) it may fail to compile, or (2) it may fail to match the user's expectations. In the first case, we can automatically pass it back into Codex along with the prompt "Fix" and the error traceback from the compiler, and the technical errors will be ironed out. In the second case, the user may specify a fix or modification prompt and feed it into Codex to try and bring the output image closer to their expectations. In our example, the user has decided to make the cube blue, and the suggested prompt makes this desired modification. This iterative process allows users to easily produce images that are precisely specified and match their desired initial prompts. Performance on our benchmark demonstrates the efficacy of the approach.

On the bottom row, a user prompt to change the cube color results in changes in the code that achieve the desired modified rendering.

Our study is motivated by: (1) the ability of a creative person without a computer-aided design background to generate a precise rendering of an object, matching their text prompt; (2) an investigation into the limits of current text-to-image models to accomplish (1). We observe that current text-to-image algorithms struggle for tasks requiring precision, counting, specific geometric shapes, and sequential modifications of an existing object. To overcome these limitations, we leverage text-to-code to add a measure of precision to the creative process, allowing precise numbers, shapes, and objects while preserving the creative flexibility of text-to-image prompts. The precision enabled by code affords three distinct classes of the image to be generated: (1) instances where specific shapes are required; (2) instances where specific numbers of objects are needed; (3) instances where a pre-existing object or code rubric is modified intentionally, for example, to create a time-series of objects responding to an applied force in a physics-based simulation, or sequential modifications to an object to add complexity or modify it according to user specifications.

3. Results

3.1. Text to Graphics

We prompted GPT-4 to create an animated Rubik's cube using SVG in JavaScript. Running the generated code rendered a 3-D colored Rubik's cube with rotation functionality. Next, we gave GPT-4 input to create an animated model of the Earth with continents and oceans using a texture loaded locally. Running the generated code rendered a 3-D model of the Earth rotating on its axis, using a texture loaded from local files. Since vanilla JavaScript lacks the functionality to support complex graphics, we prompted GPT-4 to use the Three.js library and modify a complex scene. Figure 1 demonstrates GPT-4's ability to understand over 250 lines of complex graphics code and correctly modify the code to increase the size of the mirror ball.

3.2. Error Correction and Modification

We take the Cornell box, and modify it according to user prompts. We illustrate seven examples out of 100 attempts in Figure 3. We evaluate the success of our workflow in two different ways: (1) whether the image was correctly modified according to our instructions; (2) whether the image-generation code compiled on the first iteration. The results for (1) are shown in Table 2, indicating a 65% success rate. 91% of the generated codes compiled with error correction. A key take-away, is that this workflow has a high success rate modifying objects within 3D renderings.

3.3. Procedural Rendering

We explore the creation of a three-dimensional object using recursion. Three different algorithms are compared: Stable Diffusion, DALL·E 2, and our rendering algorithm. The results are shown in Figure 4, with a prompt for the fractal tetrahedron. In Figure 4(d), we prompt our algorithm to change the object's color and background. The alignment between Figures 4(c) and 4(d) and the original text prompt (a fractal tetrahedron) is self-evident.

As a third example, we consider the task of procedural rendering, which is nearly impossible to accomplish using stable diffusion and DALL·E 2. Procedural rendering consists of progressive modification of an existing object, increasing the level of detail and complexity with each rendering. Figure 5 illustrates three spaceships created in blender using procedural rendering. In the case of trees, only the parameters were modified; here, the code and the parameters are modified to achieve the diversity of spaceships shown.

3.4. Precise Rendering

We render a precise number of objects. A common shortcoming of Stable Diffusion and DALL·E 2, is the inability to render a precise number of objects when prompted by a user. In this case, we illustrate using a bouquet of a precise number of Phyllotaxis flowers. As shown in 6, both Stable Diffusion and DALL·E 2 generate more flowers than the user prompt. As a further illustration of the ability to modify images, once rendered, Figure 6(d) shows the change of the flower color from yellow to red.

3.5. Parametric Rendering

We consider the case of code rendering a system of parametric equations describing a torus. In Figure 7, we attempted to run an outdated Blender code (describing a torus) on the latest blender version, resulting in an error. The error correction feature in our workflow corrected this error and produced the proper rendering of the torus. We then used the text prompt to add a cylinder in the center of the torus; the result is shown in Figure 7(f). These results from our workflow are compared to text prompts of a "parametric torus" in Stable Diffusion and DALL·E 2, with and without the cylinder in the middle.

As a sixth example shown in Figure 8, we illustrate the ability of our workflow to render a new genus of trees based on a set of variables and rules describing trees. Our training set consists of a dataset of 14 trees, available on GitHub [1]. These 14 trees are each created by the same 34 lines of code but with different variables (90 in total). After training our workflow on the 14 examples of a tree (few-shot learning), we issue the text prompt for "Sequoia." Program synthesis

[1] https://github.com/friggog/tree-gen

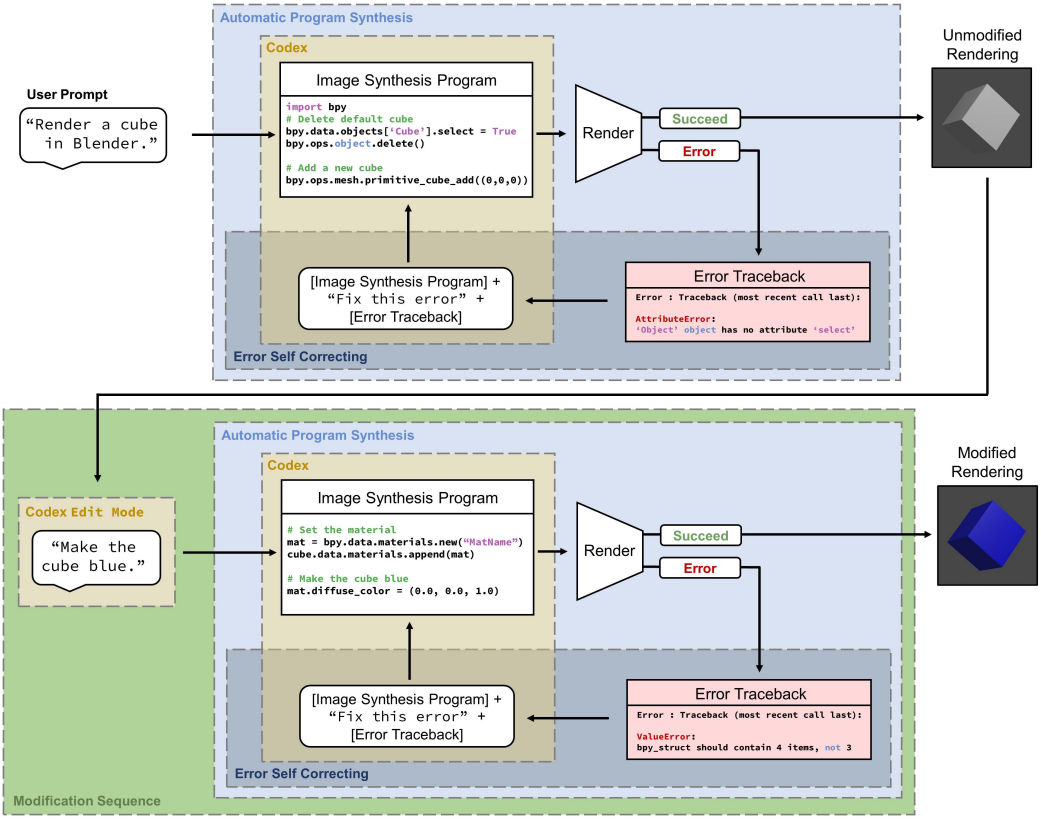

Figure 2. Rendering program synthesis with automatic error correction workflow. A user prompts Codex to generate a Blender rendering, generating an image synthesis program. At zero shot, this rendering program generated by Codex may succeed or fail due to error. If the program succeeds, the loop ends. If the program fails, the error traceback is fed back into Codex edit mode with the prompt to "fix" the error. This error correcting loop automatically repeats until all errors are corrected and a rendering is successfully output. Once the rendering is successful, the user may modify output further, *e.g.,* change the object's color. This user-modification triggers the program synthesis and error self correcting subroutines iteratively until the new modification of the rendering is successfully output.

Category	Score
Runs and renders correct content	66/100
Runs and renders incorrect content	25/100
Does not run and ends with an error	9/100

Table 2. We render the Cornell box using OpenGL by program synthesis a hundred times, each time performing a single modification. We tally the number of programs that run, render the correct output, and those that do not run.

creates the corresponding code and variables, which result in Figure 8(e).

3.6. Rendering Material Synthesis

We demonstrate the modification of an object's material constitution according to a text prompt. Figure 9 illustrates the original rendering in (a), and modified images in (b), (c) and (d). Note the changes of transmissivity, diffuse and specular reflectance, shading, and color.

3.7. Physical Simulation

We consider modifying a time series of a physics-based simulation comprising many interacting small bodies. Figure 10 shows the original simulation in the top row, consisting of a stack of cubes that are then released and allowed to interact and collide. The bottom row shows a modification created using a text prompt and our workflow. The prompt instructs our workflow to modify the simulation as follows: reduce the edge of the cube stack by half and increase gravitational acceleration by 3x. Representative screenshots of

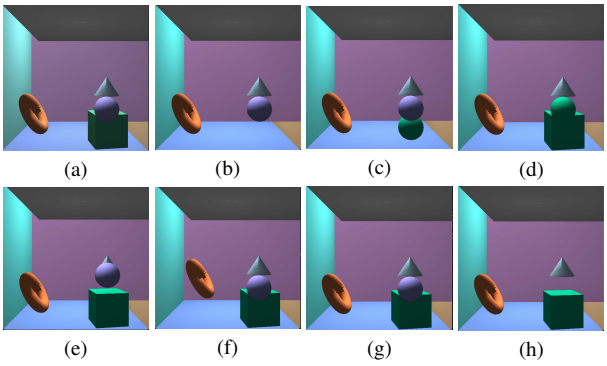

Figure 3. Cornell box: (a) Original rendering of Cornell box (original). (b) Prompt: Remove the cube (correct). (c) Prompt: Replace cube with sphere (correct). (d) Prompt: Change the sphere color to green (correct). (e) Prompt: Shift the sphere up and forward (correct). (f) Prompt: Add another white sphere above the cube (incorrect). (g) Prompt: Insert a cuboid (incorrect). (h) Prompt: Change the sphere to a triangular pyramid (incorrect).

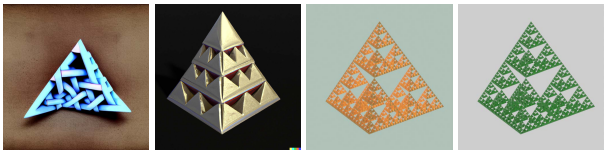

Figure 4. Fractal. Procedural recursive result: A fractal tetrahedron generated by (a) Stable Diffusion (Prompt: a fractal tetrahedron), (b) DALL-E 2 (Prompt: a fractal tetrahedron), (c) Rendering of original Blender code (d) Program synthesis edit (Prompt: change tetrahedron and background colors).

Figure 5. Spaceships. Procedural rendering of spaceships using Blender.

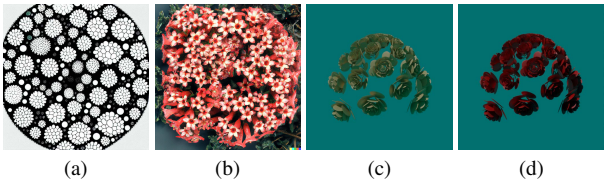

Figure 6. Flowers. A precise number of objects: 24 Phyllotaxis flowers generated by (a) Stable Diffusion (Prompt: 24 phyllotaxis flowers), (b) DALL-E 2 (Prompt: 24 phyllotaxis flowers), (c) Program synthesis edit (Correction: fix error trace) (d) Program synthesis edit (Prompt: change flower color).

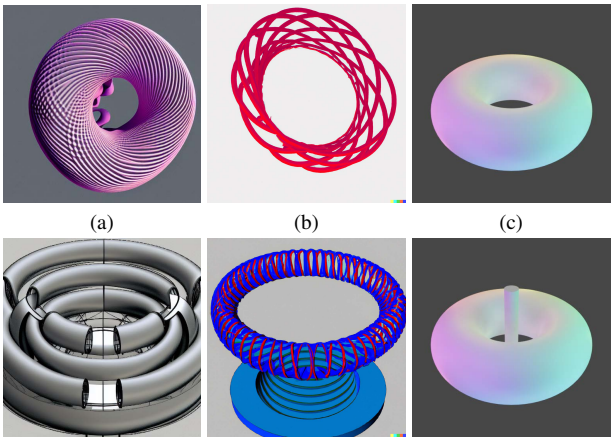

Figure 7. Parametric rendering. A parametric object and adding an object: a parametric torus generated by (a) Stable Diffusion, (b) DALL-E 2, (c) Program synthesis with error correction. Adding a cylinder in the middle (d) Stable Diffusion, (e) DALL-E 2, (f) Program synthesis.

the resulting time-series of the interacting bodies is shown on the bottom row.

3.8. Graphics Stories

We illustrate our workflow by generating a thousand results, which ties together the numerous advantages described in the text's main body— precision, procedural evolution, and error correction. We write a script that allows us to create a meta-language. This meta-language generates 1,000 simple graphic stories. Each graphic story has 2–5 prompts that specify changes to the last 3D scene. Examples of prompts are "add a green pentagrammic prism," "Add a bright hexagonal pyramid," "Move green pentagrammic prism (6,7) inches left", "rotate green pentagrammic prism 150 degrees counter-clockwise", "scale creen gentagrammic prism by 0.5". Each graphic story builds toward the final 3D scene by sequentially following the instructions from the 2–5 prompts. We use each prompt to generate code and, in turn, execute each code which results in a rendered image or an error trace. If there is an error trace, then the program and the error trace are fed back into program synthesis for error correction using the prefix "fix the error." Table 3 summarizes the statistics of the 1,000 story runs.

3.9. Photorealistic Driving Simulation

We modify the CARLA driving simulator [8], which runs on top of the Unreal 5 Engine [10] and is controlled by Python scripts. We use text-2-graphics program synthesis to modify traffic and environmental conditions. Figure 11a shows a rendered base image of traffic in a street. Given the

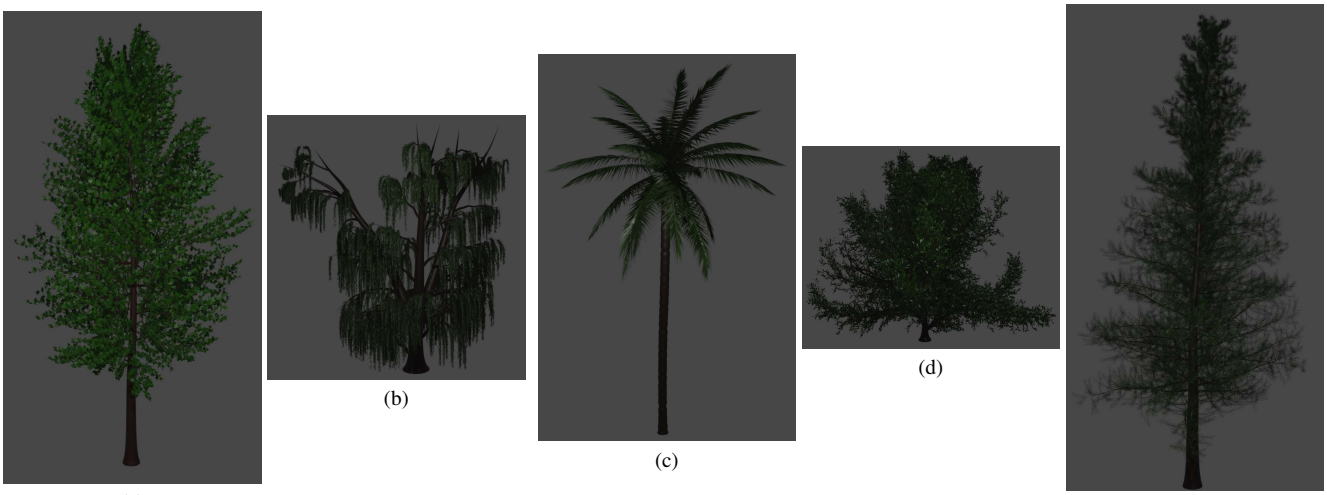

Figure 8. Parametric trees generated using few-shot learning of parameters: Procedural rendering of trees (a) Apple. (b) Willow. (c) Palm. (d) Cambridge Oak, and (e) Program synthesis generating the parameters of a Sequoia tree.

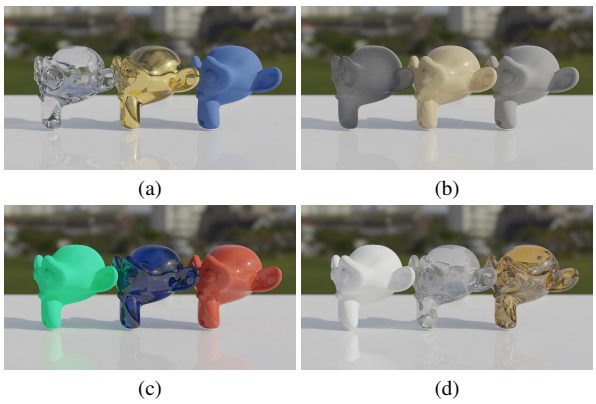

Figure 9. Rendering materials synthesis. (a) Rendering of original code: Glass, gold, blue, (b) Program synthesis: Paper, leather, stone, (c) Program synthesis: Emerald, sapphire, ruby, (d) Program synthesis: Concrete, china, diamond.

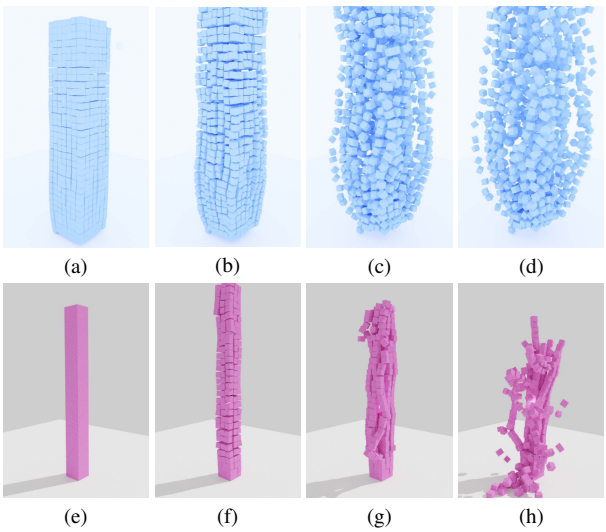

Figure 10. Physical simulation: (a-d) Original simulation (e-h) Program synthesis edit: reduce the edge of the cube stack by half and increase gravity three fold.

prompt 'Make all the vehicles red." our approach modifies Python scripts that control the simulator and renders the image shown in Figure 11b so that all vehicles are red. Notice that the graphics simulation continues to run in time; therefore, the red vehicles have different positions in a future time frame. Figures 12a and 12b demonstrate an additional text-2-graphics result. The top figure shows a base traffic image rendered using CARLA simulator on top of the Unreal 5 engine. The bottom image shows the result of text-2-graphics program synthesis given the prompt "Always have storm weather with 100% cloudy and 100% rainy."

We demonstrate text-2-graphics' precision, error correction, and procedural capabilities with over 1,000 results and detailed statistics.

3.10. Text to Photorealistic Avatars

We use structured prompts for generating a notes from questions and answers in a Machine Learning class. We generate a list of relevant topics in the course, by automatically synthesizing a prompt by sampling the questions and solutions from the problem sets, midterm exams, and final exams of the course. Each of these topics comprises a chapter in lecture notes. We use ChatGPT to turn sections into video scripts with facial gestures that drive a photo-realistic graphics avatar [25] as shown in Figure 13.

Category	Score
Rendered image	91%
Rendered image and satisfied correct user intent	63%
Did not render image	13%
Did not render but fixed by error correction	4%
Did not render and not fixed by error correction	9%

Table 3. Statistics of 1,000 text-2-graphics story results. Each story consists of 2-5 prompts. 91% of the synthesized programs generated rendered an image. 63% rendered an image matching the user intent. 13% did not render image and running the synthesized grahics program resulted in an error. 4% of these cases the program did not render but was fixed by automatic error correction. In 9% of the cases the code did not render an image and not fixed by error correction.

(a)

(b)

Figure 11. (a) Base traffic image rendered using CARLA simulator on top of Unreal 5 engine. (b) Text-2-graphics program synthesis given the prompt "Make all vehicles red." Notice that the graphics simulation continues to run in time; therefore, the red vehicles have different positions in a future time frame.

(a)

(b)

Figure 12. (a) Base weather image rendered using CARLA simulator on top of Unreal 5 engine. (b) Text-2-graphics program synthesis given the prompt "Always have storm weather with 100% cloudy and 100% rainy."

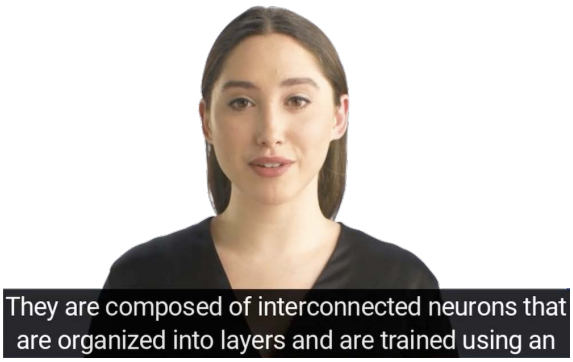

Figure 13. Rendering of an avatar given an automatically generated video script of a lecture generated using ChatGPT.

4. Conclusions

In this work, we explored the limitations of text-to-image models by creating a new multi-task benchmark with three difficulty levels. We then found tasks on which Stable Diffusion and DALLE-2 failed and provided an alternative in program synthesis. We noticed that program synthesis might generate programs with errors and performed automatic error correction using trace and program synthesis. Next, we evaluated the ability of program synthesis to modify running programs according to human intent. We demonstrate that 91% of a hundred modifications on the Cornell box render and 66% align with human intent. Finally, we demonstrate the advantages of program synthesis with error correction for text-to-graphics tasks that require precision, involve procedural renderings, and physical simulation. We can feed text-to-graphics outputs into text-to-image models for conditional image generation. Text-to-graphics allows to render virtual worlds and learn by interacting with an environment.

References

[1] Omer Bar-Tal, Dolev Ofri-Amar, Rafail Fridman, Yoni Kasten, and Tali Dekel. Text2live: Text-driven layered image and video editing. *arXiv preprint arXiv:2204.02491*, 2022. 2

[2] Brian Danchilla . Three.js framework. 2

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 2

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. 1

[5] Jaemin Cho, Abhay Zala, and Mohit Bansal. Dall-eval: Probing the reasoning skills and social biases of text-to-image generative transformers. 2022. 2

[6] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022. 2

[7] Blender Online Community. Blender - a 3d modelling and rendering package, 2018. 1

[8] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. 6

[9] Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda Chen, Sunny Tran, Newman Cheng, Roman Wang, Nikhil Singh, Taylor L. Patti, Jayson Lynch, Avi Shporer, Nakul Verma, Eugene Wu, and Gilbert Strang. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32), 2022. 2

[10] Epic Games. Unreal engine 5. 6

[11] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion. *arXiv preprint arXiv:2208.01618*, 2022. 2

[12] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-prompt image editing with cross attention control. *arXiv preprint arXiv:2208.01626*, 2022. 2

[13] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022. 2

[14] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *arXiv preprint arXiv:2203.07814*, 2022. 2

[15] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, volume 8693 of *Lecture Notes in Computer Science*, pages 740–755. Springer, 2014. 2

[16] OpenAI. GPT-4 technical report, 2023. 1

[17] Jordi Pont-Tuset, Jasper R. R. Uijlings, Soravit Changpinyo, Radu Soricut, and Vittorio Ferrari. Connecting vision and language with localized narratives. *CoRR*, abs/1912.03098, 2019. 2

[18] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *Image*, 2:T2, 2021. 2

[19] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021. 2

[20] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022. 1

[21] Nicolás Rivero. The best examples of dall-e 2's strange, beautiful ai art. *Quartz*. 1

[22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022. 1

[23] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. DreamBooth: Fine tuning text-to-image diffusion models for subject-driven generation. *arXiv preprint arXiv:2208.12242*, 2022. 2

[24] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022. 2

[25] Synthesia. Avatars. https://www.synthesia.io/, 2023. 7

[26] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Self-consistency improves chain

of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022. 2

[27] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022. 2

[28] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, et al. Scaling autoregressive models for content-rich text-to-image generation. *arXiv preprint arXiv:2206.10789*, 2022. 2

[29] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, Ben Hutchinson, Wei Han, Zarana Parekh, Xin Li, Han Zhang, Jason Baldridge, and Yonghui Wu. Scaling autoregressive models for content-rich text-to-image generation, 2022. 2

[30] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022. 2