HARNESSING GENERATIVE AI: EXPLORING THE SYNERGY OF CHATGPT AND GRASSHOPPER FOR ROOM ACOUSTIC DESIGN

S Su Cundall, Edinburgh, UK

1 INTRODUCTION

In 2023, driven by OpenAI's ChatGPT, generative AI rooted in Large Language Models (LLMs) has risen to global prominence. Functioning through natural language prompts, it adeptly handles a multitude of tasks, from responding to queries and enhancing texts to generating code. The genuine worth of AI shines when applied to real-world challenges. Professionals across various domains are now actively exploring its potential.

This paper delves into the realm of harnessing generative AI for acoustic design, specifically by integrating its code generation capabilities with existing software. The focus is the synergy of OpenAI's generative pre-trained transformer (GPT) models, with a particular emphasis on ChatGPT, and Grasshopper - a parametric coding plugin within Rhino. A case study is employed to examine the application of OpenAI's GPTs for coding within Grasshopper for sound reflection analysis.

The primary goal is to explore potential of ChatGPT and generative AI in general as a valuable tool for acoustics design and inspire further research and development in this field.

2 BACKGROUND

2.1 Significance of exponential growth in technology

Led by ChatGPT, we are witnessing a remarkable surge of generative AI tools to the public domain in 2023. What once seemed like science fiction is now a reality, and the rapid pace of iterations is astounding. Much information I collected in May for this paper is already outdated.

To grasp the significance of this recent progress, it’s essential to take a broader view of the history of technology. In their 2014 bestseller "The Second Machine Age"[1], Brynjolfsson and McAfee divided technology history into three phases: the pre-1st machine age, the 1st machine age driven by combustion and electric engines, and the 2nd machine age powered by digital technology. As Figure 1 illustrates, when technology enters a new era, its growth curve steepens. Currently, we find ourselves at an inflection point in the 2nd Machine Age, transitioning from the internet era to the AI era.

Brynjolfsson and McAfee highlighted that the exponential improvement in computational power and digital technology's various components would primarily manifest in AI technologies. Exponential growth's significance cannot be overstated. It is akin to the famous chessboard story in which the inventor requested the king to award him rice by placing a single grain of rice on the first square and doubling the amount on each subsequent square. By the time they reached the halfway point, the required rice exceeded the entire kingdom's resources. Nine years has passed since the book's publication, and we are likely reaching the halfway mark on this metaphorical chessboard.

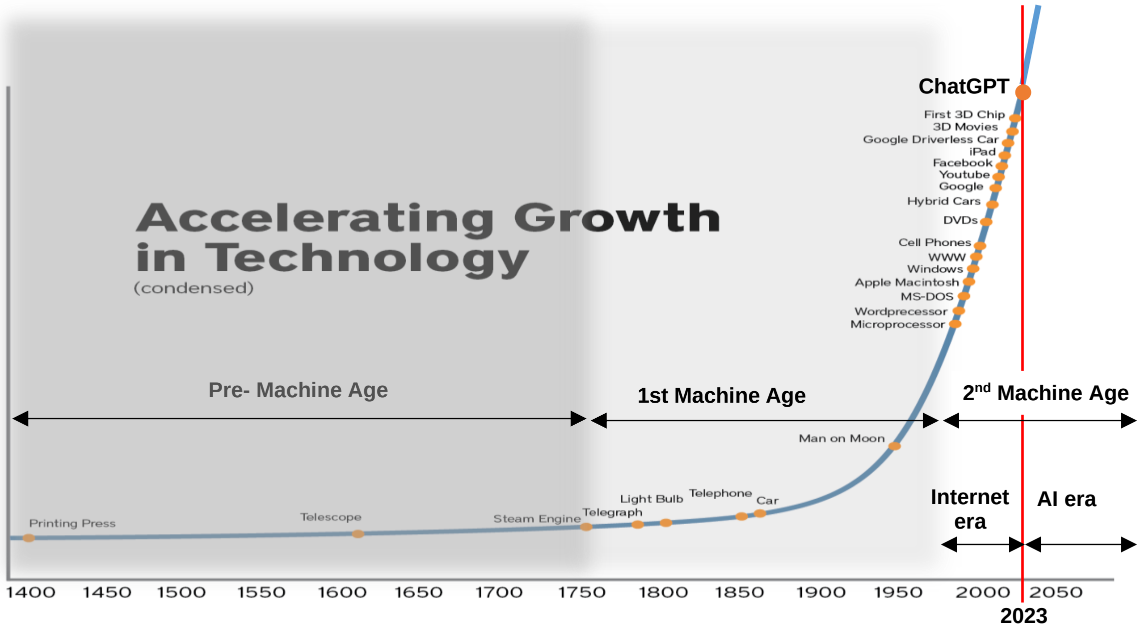


Figure 1. Accelerating growth in technology (modified from the figure in Komerath's paper[2])

The McKinsey Global Institute[3] projects that by 2030, approximately 70% of companies will adopt some form of AI technology, with nearly half of large companies implementing a full range of AI solutions. McKinsey also estimates that AI could contribute an additional $US 13 trillion in economic output by 2030, thereby increasing global GDP by approximately 1.2% annually.

How can AI technology generate value in the field of acoustics, and how can acoustic professionals make the most effective use of AI?

2.2 A practical path of harnessing AI for acoustic design: through coding

A practical approach involves leveraging generative AI programming applications based on Large Language Models (LLMs), such as OpenAI's ChatGPT (specifically GPT-3.5 and GPT-4), GitHub Copilot fine-tuned with GPT-3, Google's Bard powered by PaLM-2, Amazon's CodeWhisperer, Meta's LLaMA, Anthropic's Claude-2, and Hugging Face and Big Science's Bloom. These LLM applications are robust, capable of generating code from natural language prompts, a process known as "prompt engineering." Integrating AI-generated code into existing applications presents a potential efficient pathway to harness AI capabilities, which is the primary focus of this paper.

Figure 2 illustrates this process analogously to the distribution of electric power for electrical appliances. LLMs function as power stations, with their parameter size directly reflecting the digital power they can store. This digital power is transmitted via internet infrastructure to reach broad users, then harnessed for code generation through prompt engineering, and subsequently applied to current software applications.

To harness AI generated code, it is crucial for these software applications to include a coding interface, akin to sockets and plugs for appliances to access electricity from power stations. Applications like Rhino with Grasshopper, Revit with Dynamo, and Excel-based add-ins fall into this category, providing a convenient means of harnessing AI-generated code. However, other applications such as Odeon, SoundPlan, and Insul do not possess such a component. To utilise AI power with LLMs, they need either internalise the interface or integrate a coding component. Otherwise, newer applications powered by LLMs may swiftly replace them.

As of the release of this paper, OpenAI's GPT-4 stands out as the uncontested leader in coding abilities[4], and ChatGPT earns favour among developers as depicted in Figure 3[5]. Therefore, this paper focuses on the study of OpenAI's GPTs with a particular emphasis on ChatGPT.

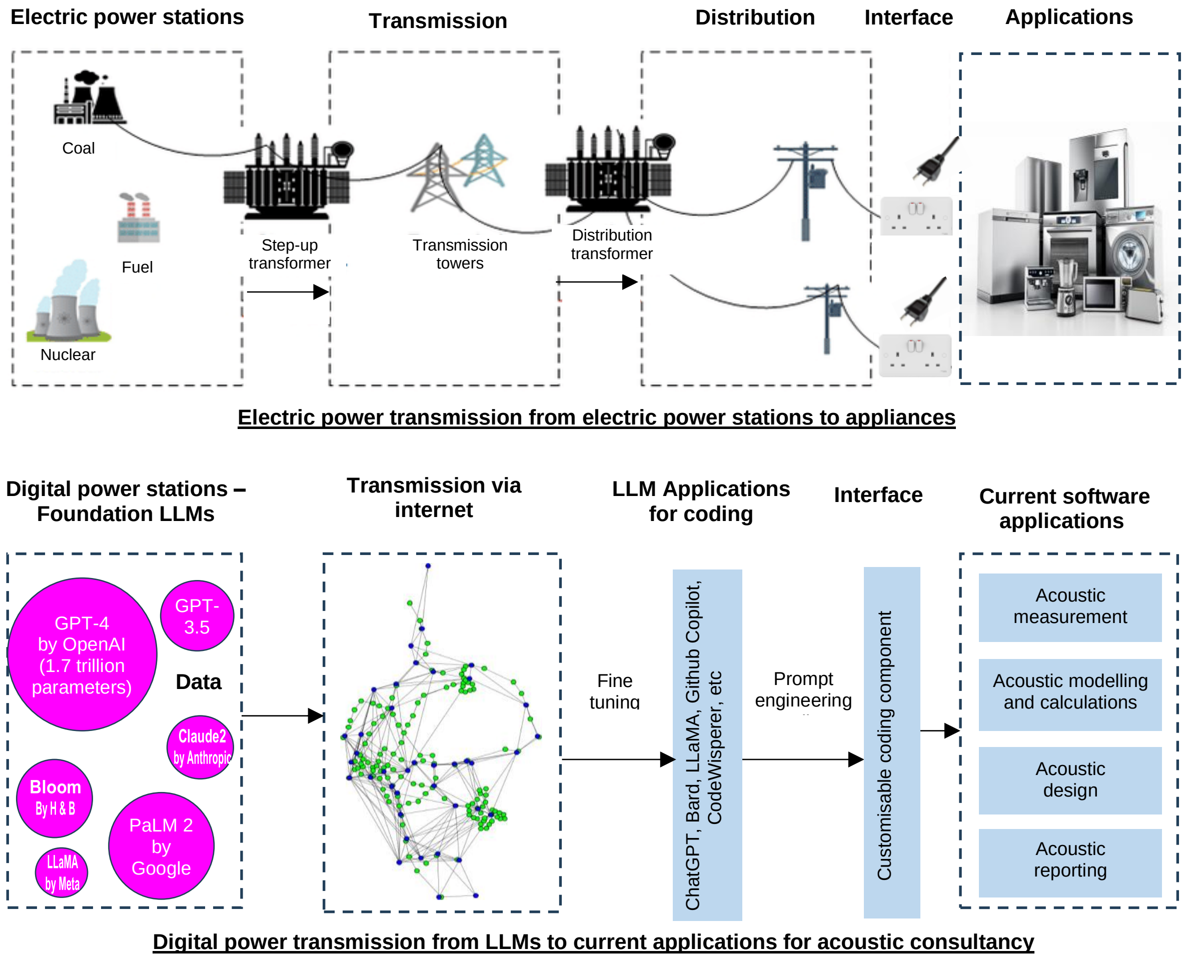


Figure 2. An analogue of transmission of digital power to existing applications and electric power to appliances (The sizes of the LLMs are based on Wikipedia[6] and Khawaja[7])

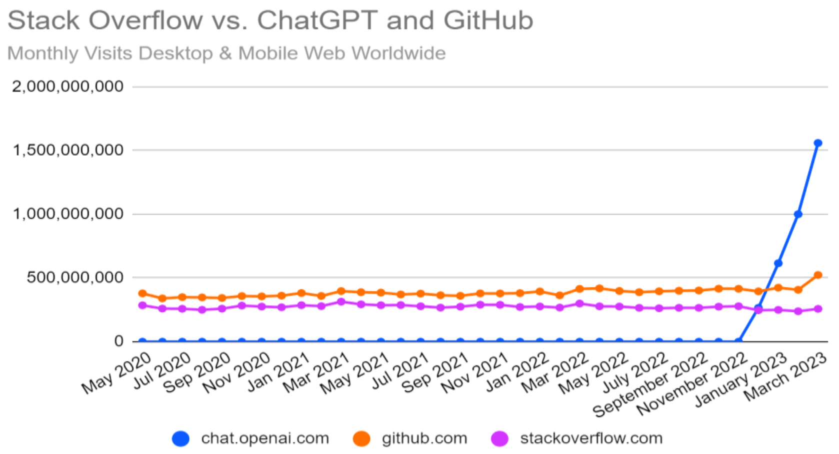


Figure 3. Comparison of monthly visits worldwide of Stack Overflow (a pre-AI de facto source for developer), ChatGPT and GitHub[5]

2.3 Grasshopper for Rhino

Grasshopper serves as a visual programming language and graphical algorithm editor plugin for Rhino, a widely used 3D computer-aided design software. It incorporates three types of programming nodes that support scripting using Python, C# and Visual Basic respectively, which can act as a bridge to harness the power of generative AI, such as OpenAI's GPT applications.

While various 3D room acoustic modelling software like Odeon, EASE, and CATT-acoustic are valuable for predicting room acoustic parameters, they often lack the flexibility required to interact with architectural models. This limitation makes them time-consuming for studying design iterations aiming at optimising designs. In contrast, Grasshopper excels in parametric design and can seamlessly integrates with Rhino, allowing users to create models that can be easily adjusted by modifying parameters in real-time within the Rhino model. This feature makes Grasshopper a potent tool for exploring diverse design variations and iterations.

2.4 Case study of the synergy of OpenAI's GPT and Grasshopper

In the realm of room acoustics for performing arts venues, optimising the angles of reflectors for early reflections holds great importance but can be time-consuming. The following section presents a case study exploring the collaboration between OpenAI's GPT model and Grasshopper in optimising reflector angles to maximise the projection of 1st order reflection energy onto the audience plane. The Python node within Grasshopper acts as the interface between the two applications.

3 OPENAI'S CODE GENERATION METHODS AND PROMPT ENGINEERING

OpenAI offers multiple options for code generation. This section provides an overview of these methods and how they were implemented for the specific tasks for the case study in this paper.

3.1 Use ChatGPT

ChatGPT is built upon either GPT-3.5 or GPT-4 - members of OpenAI's proprietary series of GPT models, and is fine-tuned for conversational applications using a combination of supervised and reinforcement learning techniques[8]. It can generate code and coding guidance by inputting prompts that define coding tasks directly within its contextual window.

In July 2023, ChatGPT introduced a novel feature known as "Custom Instructions," which encompasses two questions as below aimed at refining its responses. For the task of interest, the following custom instructions guided by online examples[9] have been set up to investigate whether they can improve the quality of the outputs.

What would like ChatGPT to know about you to provide better responses?

- *Profession: I am a leading acoustic expert specialising in room acoustic design.*
- *Key responsibilities: My primary responsibility involves writing Python code for Grasshopper, a modelling and scripting plugin for Rhino 3D.*
- *Knowledge or expertise: I have extensive knowledge and expertise in the field of acoustics, Python programming, and utilizing Grasshopper for design purposes.*
- *Typical challenges: I frequently encounter challenges related to writing Python code specifically tailored for room acoustic design within the Grasshopper environment.*
- *Current projects: I am currently engaged in optimizing the design of reflectors to enhance acoustic performance.*
- *Jargon or Terminology: My work involves using terms such as "1st order reflection," "omni-directional source," "reflector," and "audience plane" within the context of acoustics and design.*

- *Goals and objectives: My primary goal is to automate the process of acoustic design to improve efficiency and precision in my work.*
- *Interactions: I frequently collaborate and interact with other acoustic professionals to exchange ideas and knowledge within the field.*

How would like ChatGPT to respond?

- *Tone and Formality: instructive*
- *Level of detail: detailed guidance on code writing*
- *Preferred references: Rhino and Grasshopper user manuals and online forums*
- *Examples or analogies: function examples and syntax explanations*
- *Avoidance of ambiguity: clear and direct guidance*
- *Resource links: OpenAI and Rhino website links*
- *Problem solving method: Step-by-step guidance on code writing guidance*

ChatGPT based on GPT-3.5 is free but ChatGPT based on GPT-4 costs $US 20 per month excluding VAT.

3.2 Use OpenAI's Playground

OpenAI's Playground, featuring a system prompt window, a user prompt window, and a configuration panel, is another option for code generation.

The system prompt resembles ChatGPT's "Custom Instructions" and the user message window is akin to ChatGPT's chat window. It was known LLMs are great prompt engineers[4]. The prompt-for-prompt technique, which uses a prompt which is the output of another prompt, was employed to generate the system prompt for the task (as shown in Figure 4), using the output of the prompt "*How would you describe a Grasshopper Python expert for acoustic design?*".

The Playground allows the configuration of inference parameters, such as Maximum New Token, Temperature, Top P, Stop Sequences, Frequency Penalty, and Presence Penalty. LLMs predict the next word from the previous one using random sampling. While Maximum New Token controls the total length of the input and output, all other parameters influence randomness. Temperature, ranging from 0 to 2, is most useful, adjusting the weighting of probability distribution for next word prediction. A lower temperature results in less randomness of output, while a higher temperature yields more creative output. For code generation, a temperature setting of 0 is preferable to ensure a deterministic output.

The Playground also enables the selection of OpenAI's LLMs. Playground is charged according to the usage of tokens. Note that GPT-4 costs 10 times of GPT-3.5-turbo-16k.

3.3 Use fine-tuning on OpenAI's platform

If the aforementioned methods fail to yield satisfactory code, OpenAI's platform offers a fine-tuning function to enhance output quality. This process doesn't necessitate extensive coding; rather, it relies on a low-code method. It involves in building a dataset including examples of prompts along with manually created code that corresponds to these prompts. By doing so, the platform fine-tunes the chosen LLM. Nevertheless, this method requires a substantial number of quality examples to be effective. Regrettably, time constraints have precluded its exploration for the case study. However, it remains a potential avenue for future investigation.

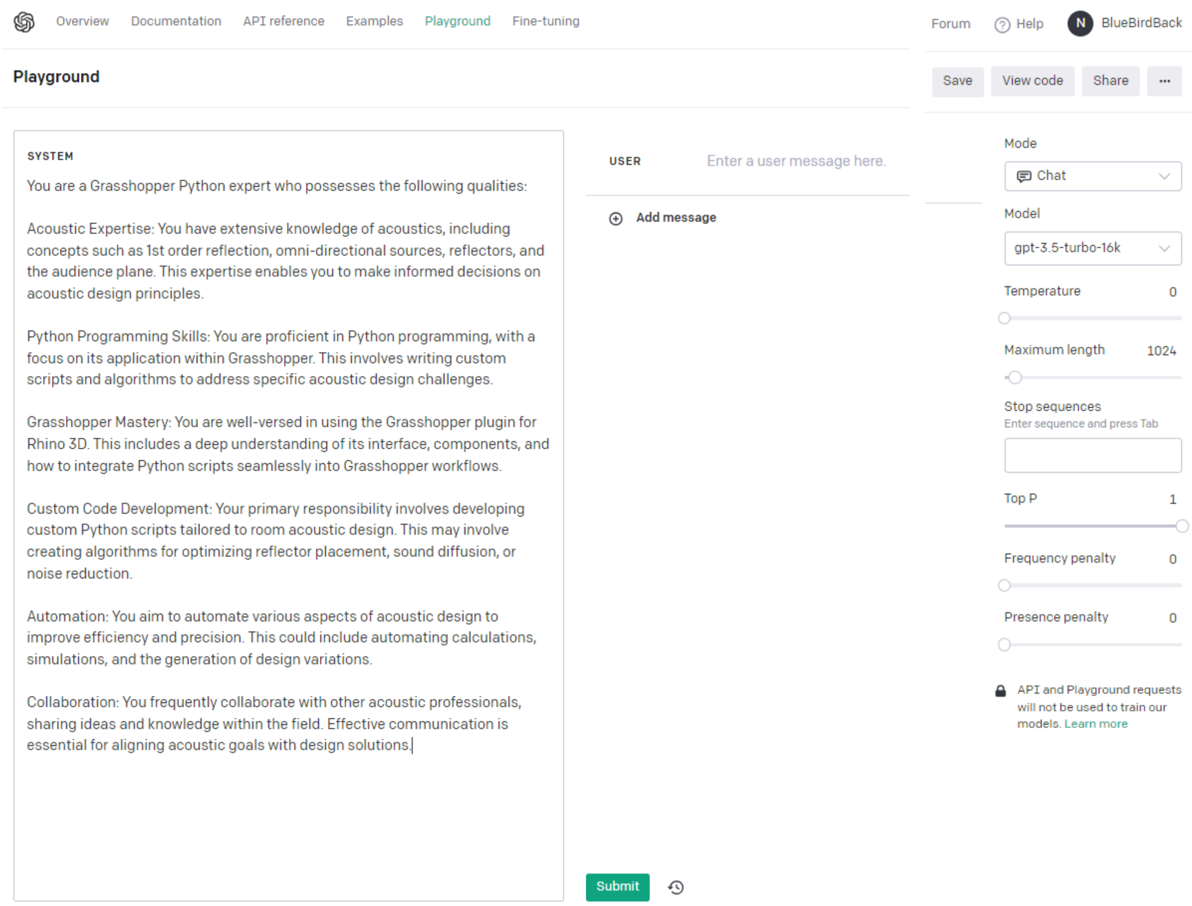


Figure 4. OpenAI Playground settings for the case study

4 THE PROCEDURE FOR OPTIMISING REFLETOR ANGLE

4.1 Crafting a coding strategy

Optimising reflector angle for 1st order reflection is a complex task, requiring a good understanding of physics, geometry, and programming. While none of OpenAI's methods can directly generate Python code to complete this task, they can serve as a valuable guide, offering a roadmap to navigate this intricate process.

The prompt below was used for generating the road map:

“Provide a step-by-step guidance on writing Python code for Grasshopper to optimise the angles of a reflector to maximise the projection of 1st order reflection energy onto the audience plane. The inputs include a reflector and its rotation axis, the location of an omni-directional source, a range of angles, and the audience plane as the receiving surface. The desired output is the angle of the reflector that projects the most energy onto the receiving plane.”

Multiple methods were tested, with the evaluations provided in the table below. It's worth noting that the Custom Instructions significantly enhanced the output quality of ChatGPT, and the comparison of the outputs with and without the Custom Instructions are included in Figure 5. GPT-4, when coupled with well-constructed prompts, further improved the output quality. However, considering ChatGPT (GPT-3.5) is a cost-free option, ChatGPT (GPT-3.5) with Custom Instructions is the recommended initial step. A significant portion of the code for the case study was generated using ChatGPT (GPT-3.5). However, none of the instructions explicitly guided the creation of an omni-directional source.

<table>
<tr><th colspan="2">ChatGPT (GPT-3.5)</th><th colspan="5">OpenAI Playground</th></tr>
<tr><td>(a) Without Custom Instructions</td><td>(b) With Custom Instructions as described in Section 3.1</td><td>(c) GPT-3.5-Turbo-16k with system prompt the same as the Custom Instructions</td><td>(d) GPT-3.5-Turbo-16k with the system prompt generated using the prompt-for-prompt technique</td><td>(e) GPT-4 with system prompt the same as the Custom Instructions</td><td>(f) GPT-4 with system prompt generated using the prompt-for-prompt technique</td><td>(g) GPT-4 with default system prompt</td></tr>
<tr><td colspan="2">The Custom Instructions improved the output quality significantly.</td><td colspan="5">The differences of the output quality are not significant.
The outputs using Methods (e) and (f) were more instructive, giving more information about the functions for achieving the tasks.</td></tr>
<tr><td colspan="7">ChatGPT (GPT-3.5) with no custom instructions produced least quality of output. GPT-4 with well designed system prompt produced the most instructive output. However, the benefit of GPT-4 was insignificant compared with GPT-3.5.</td></tr>
</table>

Table 1. Comparison of output quality using various OpenAI methods

With the guidance provided by the outputs of OpenAI methods, the following steps were undertaken to accomplish the case study:

- Creating an omni-directional sound source.
- Finding the rays received by a reflector.
- Finding the reflected rays received by the audience plane.
- Optimising the angle of the reflector to project the maximum energy onto the audience plane.

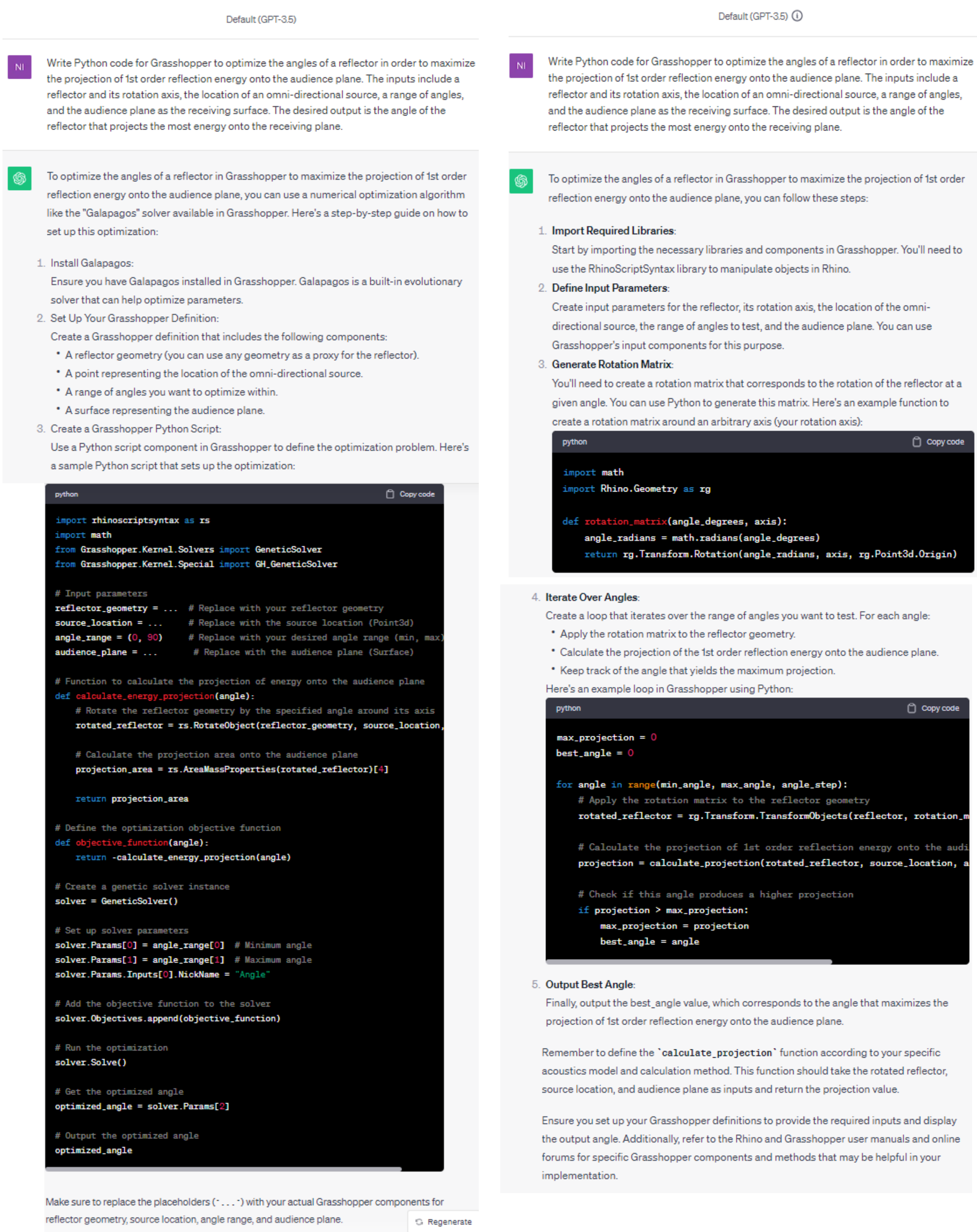
4.2 Creating an omni-directional sound source

For an omni-directional sound source represented by rays, the end point of each ray on the radiation sphere should be evenly spaced. This seemingly simple requirement is an interesting mathematical problem, as it turns out that for more than 20 points on a sphere, it is impossible to achieve perfect evenness. Several mathematical models are available for achieving even point distribution with varying perfections. In descending order of evenness, these models include the Fibonacci Sphere Sampling method, the Golden Spiral Algorithm, and the Spherical Coordinate method.

Now, an intriguing question arises: can OpenAI's GPT identify the optimal method and generate executable code without specific guidance on which mathematical approach to employ?

To address this question, three sets of experiments using various OpenAI methods were conducted, each set employing a different prompt technique:

- Using a prompt that directly describes the acoustic task.
- Using a prompt that describe the underlying mathematical procedure.
- Using the prompt-for-prompt technique to enhance the prompt.



Default (GPT-3.5)

NI Write Python code for Grasshopper to optimize the angles of a reflector in order to maximize the projection of 1st order reflection energy onto the audience plane. The inputs include a reflector and its rotation axis, the location of an omni-directional source, a range of angles, and the audience plane as the receiving surface. The desired output is the angle of the reflector that projects the most energy onto the receiving plane.

To optimize the angles of a reflector in Grasshopper to maximize the projection of 1st order reflection energy onto the audience plane, you can use a numerical optimization algorithm like the "Galapagos" solver available in Grasshopper. Here's a step-by-step guide on how to set up this optimization:

1. Install Galapagos:
   Ensure you have Galapagos installed in Grasshopper. Galapagos is a built-in evolutionary solver that can help optimize parameters.
2. Set Up Your Grasshopper Definition:
   Create a Grasshopper definition that includes the following components:
   - A reflector geometry (you can use any geometry as a proxy for the reflector).
   - A point representing the location of the omni-directional source.
   - A range of angles you want to optimize within.
   - A surface representing the audience plane.
3. Create a Grasshopper Python Script:
   Use a Python script component in Grasshopper to define the optimization problem. Here's a sample Python script that sets up the optimization:

python Copy code

```python
import rhinoscriptsyntax as rs
import math
from Grasshopper.Kernel.Solvers import GeneticSolver
from Grasshopper.Kernel.Special import GH_GeneticSolver

# Input parameters
reflector_geometry = ...  # Replace with your reflector geometry
source_location = ...     # Replace with the source location (Point3d)
angle_range = (0, 90)     # Replace with your desired angle range (min, max)
audience_plane = ...       # Replace with the audience plane (Surface)

# Function to calculate the projection of energy onto the audience plane
def calculate_energy_projection(angle):
    # Rotate the reflector geometry by the specified angle around its axis
    rotated_reflector = rs.RotateObject(reflector_geometry, source_location,

    # Calculate the projection area onto the audience plane
    projection_area = rs.AreaMassProperties(rotated_reflector)[4]

    return projection_area

# Define the optimization objective function
def objective_function(angle):
    return -calculate_energy_projection(angle)

# Create a genetic solver instance
solver = GeneticSolver()

# Set up solver parameters
solver.Params[0] = angle_range[0]  # Minimum angle
solver.Params[1] = angle_range[1]  # Maximum angle
solver.Params.Inputs[0].NickName = "Angle"

# Add the objective function to the solver
solver.Objectives.append(objective_function)

# Run the optimization
solver.Solve()

# Get the optimized angle
optimized_angle = solver.Params[2]

# Output the optimized angle
optimized_angle
```

Make sure to replace the placeholders (`...`) with your actual Grasshopper components for reflector geometry, source location, angle range, and audience plane.

Regenerate

Default (GPT-3.5)

NI Write Python code for Grasshopper to optimize the angles of a reflector in order to maximize the projection of 1st order reflection energy onto the audience plane. The inputs include a reflector and its rotation axis, the location of an omni-directional source, a range of angles, and the audience plane as the receiving surface. The desired output is the angle of the reflector that projects the most energy onto the receiving plane.

To optimize the angles of a reflector in Grasshopper to maximize the projection of 1st order reflection energy onto the audience plane, you can follow these steps:

1. **Import Required Libraries**:
   Start by importing the necessary libraries and components in Grasshopper. You'll need to use the RhinoScriptSyntax library to manipulate objects in Rhino.
2. **Define Input Parameters**:
   Create input parameters for the reflector, its rotation axis, the location of the omni-directional source, the range of angles to test, and the audience plane. You can use Grasshopper's input components for this purpose.
3. **Generate Rotation Matrix**:
   You'll need to create a rotation matrix that corresponds to the rotation of the reflector at a given angle. You can use Python to generate this matrix. Here's an example function to create a rotation matrix around an arbitrary axis (your rotation axis):

python Copy code

```python
import math
import Rhino.Geometry as rg

def rotation_matrix(angle_degrees, axis):
    angle_radians = math.radians(angle_degrees)
    return rg.Transform.Rotation(angle_radians, axis, rg.Point3d.Origin)
```

4. **Iterate Over Angles**:
   Create a loop that iterates over the range of angles you want to test. For each angle:
   - Apply the rotation matrix to the reflector geometry.
   - Calculate the projection of the 1st order reflection energy onto the audience plane.
   - Keep track of the angle that yields the maximum projection.

   Here's an example loop in Grasshopper using Python:

python Copy code

```python
max_projection = 0
best_angle = 0

for angle in range(min_angle, max_angle, angle_step):
    # Apply the rotation matrix to the reflector geometry
    rotated_reflector = rg.Transform.TransformObjects(reflector, rotation_m

    # Calculate the projection of 1st order reflection energy onto the audi
    projection = calculate_projection(rotated_reflector, source_location, a

    # Check if this angle produces a higher projection
    if projection > max_projection:
        max_projection = projection
        best_angle = angle
```

5. **Output Best Angle**:
   Finally, output the best_angle value, which corresponds to the angle that maximizes the projection of 1st order reflection energy onto the audience plane.

Remember to define the `calculate_projection` function according to your specific acoustics model and calculation method. This function should take the rotated reflector, source location, and audience plane as inputs and return the projection value.

Ensure you set up your Grasshopper definitions to provide the required inputs and display the output angle. Additionally, refer to the Rhino and Grasshopper user manuals and online forums for specific Grasshopper components and methods that may be helpful in your implementation.

(a) ChatGPT(GPT-3.5) with no Custom Instructions (b) ChatGPT(GPT-3.5) with the Custom Instructions

Figure 5. Comparison of the outputs by ChatGPT (GPT-3.5) with and without the Custom Instructions described in Section 3.1

4.2.1 Experiment 1 - Using a prompt that directly describe the acoustic task

Prompt:
"Write a Grasshopper Python function to generate an omni-directional source represented by rays. End points of rays should have equal distance to adjacent points on the radiation sphere. The Input parameters are location coordinates of the source, number of rays, radius of sphere of the rays. Output are ray vectors with lengths as defined radius."

The outputs are evaluated as below.

	ChatGPT (GPT-3.5)		**OpenAI Playground**		
OpenAI Method	(a) Without Custom Instructions	(b) With Custom Instructions as described in Section 3.1	(c) GPT-3.5-Turbo-16k with system prompt the same as the Custom Instructions	(d) GPT-4 with system prompt the same as the Custom Instructions	(e) GPT-4 with default system prompt
Output	Acceptable. Using the Spherical Coordinate method				

Table 2. Comparison of output quality using various OpenAI methods – Experiment 1

4.2.2 Experiment 2 - Using a prompt that describe the underlying mathematical procedure

Prompt:
"Grasshopper Python code: distribute points evenly on a sphere. The input parameters are co-ordinates of the centre of the sphere, number of points, radius of sphere. Output: vectors from centre of the sphere to the points on the sphere."

The outputs are evaluated as below.

	ChatGPT (GPT-3.5)		**OpenAI Playground**		
OpenAI Method	(a) Without Custom Instructions	(b) With Custom Instructions as described in Section 3.1	(c) GPT-3.5-Turbo-16k with system prompt the same as the Custom Instructions	(d) GPT-4 with system prompt the same as the Custom Instructions	(e) GPT-4 with default system prompt
Output	Acceptable. Using spherical coordinate	Better. The Golden Spiral Algorithm	Acceptable. Using spherical coordinate method	Better. The Golden Spiral Algorithm	Better. The Golden Spiral Algorithm

Table 3. Comparison of output quality using various OpenAI methods – Experiment 2

4.2.3 Experiment 3 – Using the prompt-for-prompt technique to enhance prompt

Prompt 1:
"Improve English below for better clarity: Grasshopper Python code - define a function for distributing points as even as possible on a sphere using the best algorithm. The input parameters are co-ordinates of the centre of the sphere, number of points, radius of sphere. Output: vectors from centre of the sphere to the points on the sphere. The length of vector equal to the radius of the sphere."

Output of Prompt 1:
"Grasshopper Python Code: Create a Function for Uniformly Distributing Points on a Sphere Using an Optimal Algorithm and Generate Vectors from the Sphere's Centre to These Points.
Input Parameters - Centre coordinates of the sphere; Number of points to distribute; Radius of the sphere.

Output - Vectors originating from the centre of the sphere to points on the sphere's surface; Each vector has a length equal to the sphere's radius.
This code accomplishes two tasks: it evenly distributes points on a sphere using an optimized algorithm and then creates vectors that connect the centre of the sphere to these points, with each vector having a length equal to the sphere's radius.”

Prompt 2: using the output of Prompt 1.

The outputs of Prompt 2 are evaluated as below.

	ChatGPT (GPT-3.5)		OpenAI Playground		
OpenAI Model	(a) Without custom instructions	(b) With Custom Instructions as described in Section 3.1	(c) GPT-3.5-Turbo-16k with system prompt the same as the Custom Instructions	(d) GPT-4 with system prompt the same as the Custom Instructions	(e) GPT-4 with default system prompt
Output	Acceptable. Straightforward spherical coordinate method	Best. Fibonacci Sphere Sampling	Best. Fibonacci Sphere Sampling	Better. Golden Spiral Algorithm	Better. Golden Spiral Algorithm

Table 4. Comparison of output quality using various OpenAI methods – Experiment 3

4.2.4 Findings

The experiments above highlighted the substantial impact of prompts on output quality. When using a prompt that directly described the acoustic task, GPT had to translate the acoustic task to a mathematical procedure first. Introducing this additional step resulted in the lowest quality of code. While the prompt-for-prompt technique did enhance output quality, it did not yield a discernible difference when applied to the GPT-4 model. Surprisingly, OpenAI's free service, ChatGPT (GPT-3.5), consistently delivered the most satisfactory results when incorporated the well-designed custom instruction.

The code generated by ChatGPT (GPT-3.5) was successfully integrated into a Python node in Grasshopper. The results are illustrated in Figure 6.

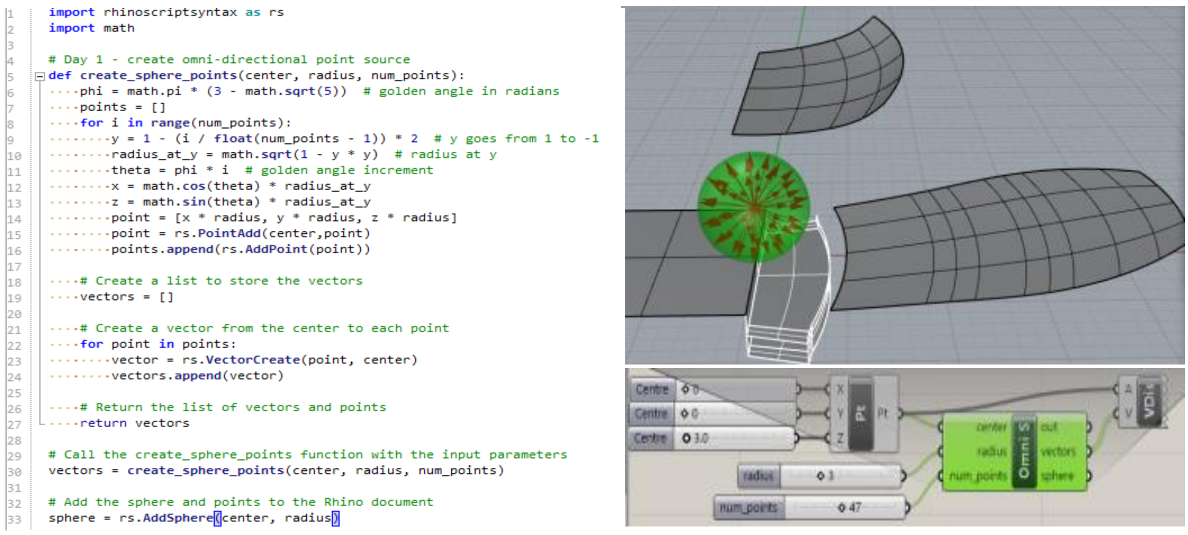


Figure 6. ChatGPT (GPT 3.5) generated Python code and its application in Grasshopper to create an omni-directional sound source

4.3 Finding rays received by a reflector

Building upon insights from previous experiment, the acoustic task of determining rays received by a reflector was translated into a mathematical problem to enhance the output quality. To ensure precision and clarity in the prompt, the prompt-for-prompt technique was employed.

The enhanced prompt for finding rays received by a reflector is as below:
Write a Python function in Grasshopper to perform the following tasks:
Find the intersection points between a set of lines and a Brep.
If there are intersection points, draw new lines from the starting points of the original lines to the intersection points.
If there are no intersection points initially, extend the lines, find the intersection points, and then draw new lines from the starting points of the original lines to these intersection points.
Output are the new lines.

Outputs generated by ChatGPT (GPT-3.5) with the Custom Instructions and Playground GPT-4 with the system prompt were remarkably similar. However, none of them was not executable. Error pertained to the functions responsible for finding the intersection between a line and a brep and extending a line. While "rs.CurveBrepIntersect" and "rs.ExtendCurve" are indeed the correct functions for these tasks (in lines 15 and 26 of the code), ChatGPT encountered issues in understanding the data type of input and syntax of these functions. Even after providing the correct syntax in the context window, ChatGPT still struggled to generate executable code.

```
7   import rhinoscriptsyntax as rs
8
9   def find_and_draw_intersection_lines(lines, brep):
10      # Initialize a list to store the new lines
11      new_lines = []
12
13      # Find intersection points between lines and Brep
14      for line in lines:
15          intersections = rs.CurveBrepIntersect(line, brep)
16
17          # If there are intersection points, draw lines from the starting points
18          if intersections:
19              for intersection in intersections:
20                  start_point = rs.CurveStartPoint(line)
21                  new_line = rs.AddLine(start_point, intersection)
22                  new_lines.append(new_line)
23
24          # If there are no intersections, extend the line, find intersections, and draw new lines
25          else:
26              extended_line = rs.ExtendCurve(line, 0, 1, rs.CurveExtensionStyle.Line)
27              intersections = rs.CurveBrepIntersect(extended_line, brep)
28              if intersections:
29                  for intersection in intersections:
30                      start_point = rs.CurveStartPoint(extended_line)
31                      new_line = rs.AddLine(start_point, intersection)
32                      new_lines.append(new_line)
33
34      return new_lines
35
36  # Call the function with your set of lines and Brep
37  resulting_lines = find_and_draw_intersection_lines(lines, brep)
```

```
utput  Help  Compile

Runtime error (TypeErrorException): iteration over non-sequence of type Guid

Traceback:
  line 14, in find_and_draw_intersection_lines, "<string>"
  line 37, in script
```

Figure 7. The non-executable Python code generated by ChatGPT (GPT-3.5) with the Custom Instructions

4.4 Finding reflected rays received by the audience plane and optimising the angle of the reflector to project most energy to the audience plane

Given that the free service ChatGPT (GPT-3.5) with the Custom Instructions proved capable of generating Python code nearly identical to the paid service GPT-4, it was employed for generating code for the tasks detailed in this section.

Again, ChatGPT fell short in providing executable code without human refinement. The recurring errors in the generated code predominantly stemmed from its lack of familiarity with the functions in the libraries of Grasshopper and Rhino. The common issues included:

- Confusion regarding the syntax of functions in Rhino.Geometry and Rhinoscrptsyntax libraries. Both libraries facilitate Python access to geometric data types and operations in Rhino, producing identical geometry, albeit through functions with different names and syntax. ChatGPT occasionally employed functions from one library but adhered to the syntax of similar functions in the other library.

- Misuse of data types for function parameters. Some functions necessitate geometric inputs into a Python node in Grasshopper as Guids, unique ID numbers referencing geometry in the Grasshopper environment. Other functions require geometry input in its native form. ChatGPT seemed unable to distinguish the two. Additionally, misused data types extended to distinguishing between lists and individual items.

Nevertheless, ChatGPT consistently excelled in formulating the logic and structure of the generated code, while mastering such tasks might appear more challenging for a machine compared to standard content such as function syntax. ChatGPT appeared to have gleaned its logic and structural knowledge from general computer programming, albeit lacking specific training in Grasshopper Python.

Furthermore, ChatGPT adeptly discerned the suitable functions and methods for tasks outlined in prompts. One of the primary challenges when working with libraries like rhinoscriptsyntax or Rhino.Geometry is ascertaining which geometric methods best suit the task at hand. ChatGPT proved invaluable in this regard. Once appropriate functions and methods were identified, their syntax could be easily verified through the autocomplete feature provided by the Python script editor in Grasshopper or by consulting available online documentation.

ChatGPT can also assist coding in the following ways:

- For improving understanding of function usage, using the prompt *"What is the purpose of this function?"*
- For debugging, employing the prompt *"What is the issue with this code?"* It is worth noting that ChatGPT proved less effective for debugging the code it generated itself, but it could be a valuable resource for debugging manually written code.
- For code optimisation, engaging with prompts like *"Simplify this code," "Refactor this code to enhance efficiency," and "Is there a more effective approach to achieve the desired outcome?"*
- For code testing, using the prompt *"Test this function using 5 cases."*
- For code documentation, employing prompts such as *"Include a detailed and high-quality docstring for the code below"* and *"Provide an explanation for this code to document its functionality."*

Leveraging the support of ChatGPT as described above, a Python node was successfully created for accomplishing the designated tasks. The produced parametric reflection analysis in illustrated in the figures 8 to 10.

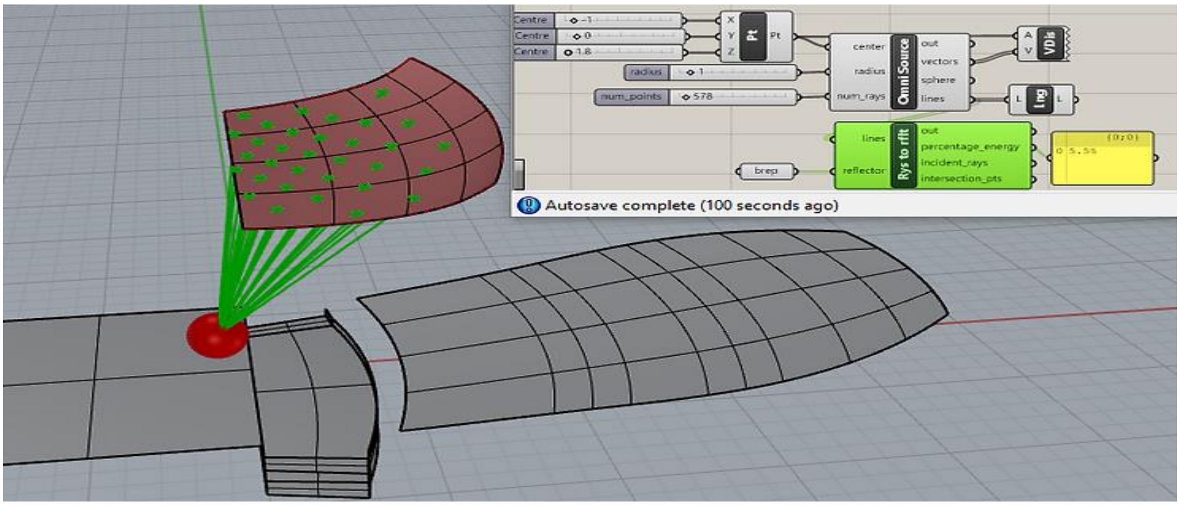


Figure 8. The parametric nodes for finding incident rays and energy distribution on a reflector from an omni-directional sound source

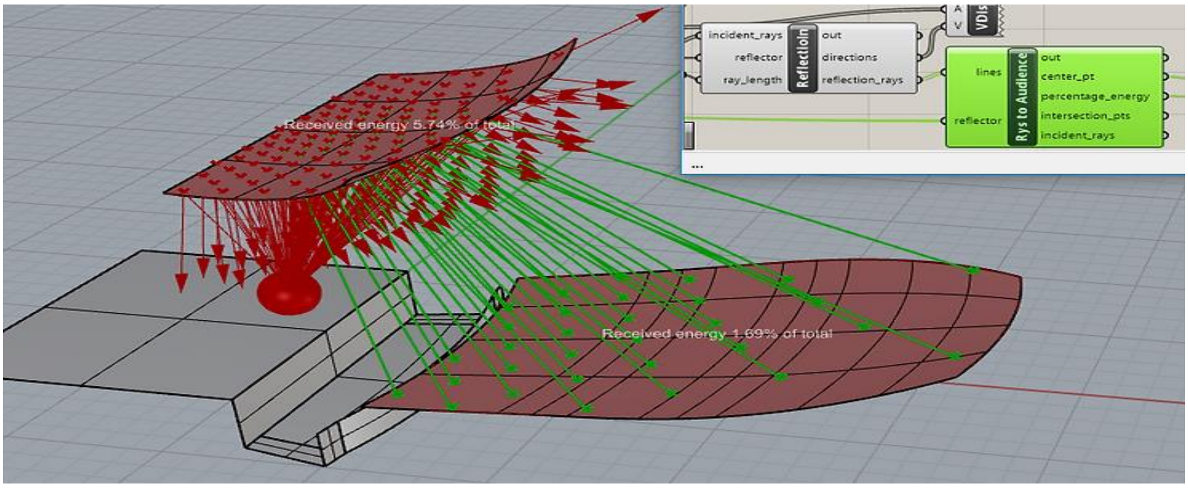


Figure 9. The parametric nodes for finding rays reflected from a reflector and energy distribution of the reflected rays onto the audience plane

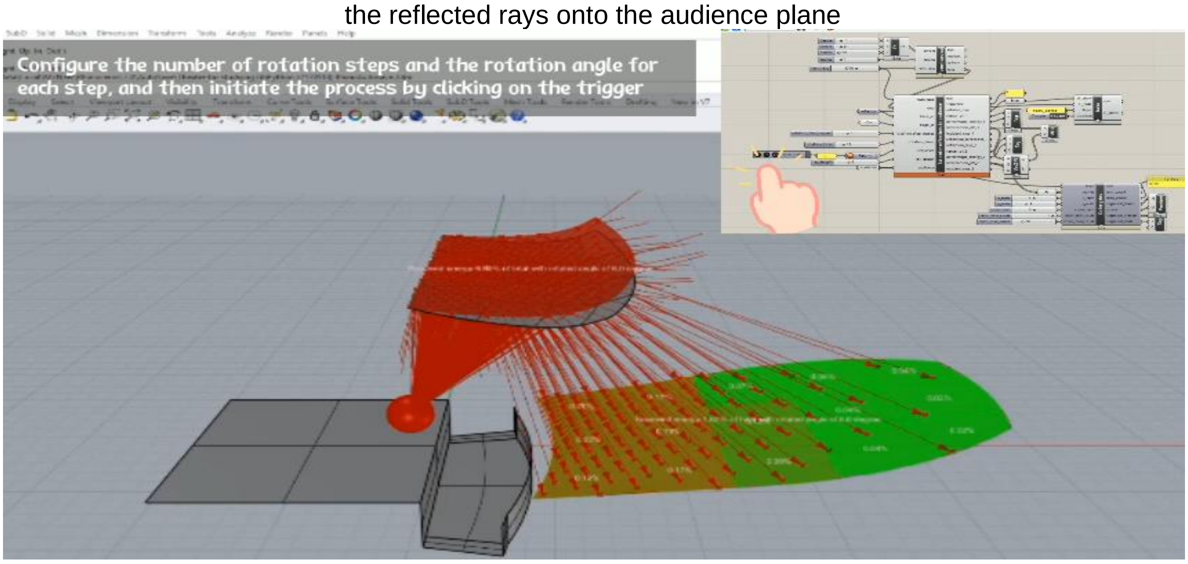


Figure10. The parametric nodes for optimising reflector angles to maximise 1st order reflection energy onto the audience plane

4.5 Discussions

The case study presented in this paper has been partially shared on my LinkedIn between April and June 2023. Various comments were attracted, which will be discussed below, along with some other discussions on using ChatGPT.

Comment one: "I'm unable to get ChatGPT to add 1 dB + 1 dB successfully......I can't imagine trying to debug an entire set of code."

This comment was received in April. At that time, ChatGPT indeed couldn't correctly perform the 1 dB plus 1 dB calculation. However, a recent attempt demonstrated that it is now capable of handling dB calculations.

Nonetheless, it's crucial to recognise that AI and humans approach problems differently. What may be straightforward for humans can be challenging for AI, and vice versa. ChatGPT's struggle with complex mathematics doesn't necessarily mean it cannot debug intricate programs if appropriately trained.

Language models like ChatGPT are primarily designed for predicting the next word based on the previous word. Consequently, they excel at processing texts but may face challenges with mathematical calculations. To address this problem, OpenAI introduced GPT-4 with external application Code Interpreter, showing remarkable performance on Math dataset with an accuracy raising from 53.9% to 84.3%[10].

Comment two: "Including examples in prompts can only improve the output for a while"

While providing examples in prompts can enhance output, ChatGPT only has a short-term memory with a capacity of 4k tokens equivalent to 3k English words. So, ChatGPT forgets any training received prior to the 3k words within the same conversation. OpenAI playground provides models which has an extended short-term memory, and the longest one is GPT-3.5-turbo-16k which has a capacity of 16k. Additionally, when a new conversation begins, ChatGPT loses all prior training. For users who wish to maintain their training, OpenAI offers a fine-tuning function on their platform as a potential solution.

Comment three: "ChatGPT confused sound pressure level with sound power level."

This could be related to the temperature parameter of ChatGPT, besides deficient training in acoustics. The temperature parameter for OpenAI's GPT, ranging from 0 to 2, governs the level of randomness in the generated output, as discussed in Section 3.3. The default value for ChatGPT is approximately 0.7. It's important to note that one cannot directly adjust the temperature of ChatGPT. However, OpenAI's Playground offers the capability to customise the temperature parameter.

Comment four: "I have found if you just give it an open-ended question, it can struggle…"

This paper has underscored the significance of prompt engineering. While it's unlikely that an open-ended question can yield useful results, even questions without an open-ended nature may still face challenges. The key lies in shortening the length of chain of thoughts, as exemplified in Section 4.2.

Canvas based visual programming vs scripts

Before generative AI became available, canvas-based programming held advantages over scripts. When knowing which node to use, one doesn't need to understand the exact syntax of the node, and the inputs and outputs are explicit. This reduces the learning curve, making it especially popular for software packages that process geometry. However, the program's format is often challenging to read, and some tasks, such as looping, can be awkward to handle. Additionally, canvas-based methods still require a significant investment of time to master, often necessitating the creation of an approach from scratch. With the assistance of ChatGPT, scripts are becoming easier to learn. Combining scripts with canvas-based visual programming allows us to leverage the best of both.

Is ChatGPT's behaviour changing over time?

The effectiveness of ChatGPT in downstream tasks is significantly influenced by the underlying LLMs. It is worthwhile to note that the same LLM version has been reported by users to have drastically different performance over time. Everyone had to continuously monitor performance as well as update carefully curated prompts. The *"How is ChatGPT's Behaviour Changing Over Time?"*[11] reveals that the March 2023 and June 2023 versions of GPT-3.5 and GPT-4 exhibited differing performances across various tasks, such as math questions, sensitive inquiries, opinion surveys, knowledge-based queries, code generation, US Medical License tests, and visual reasoning[8]. An illustration of the performance discrepancy in math questions is reproduced in Figure 11, suggesting that GPT-3.5 has seen performance improvements, while GPT-4 has experienced a decline over time, ultimately resulting in GPT-3.5 outperforming GPT-4 in June 2023.

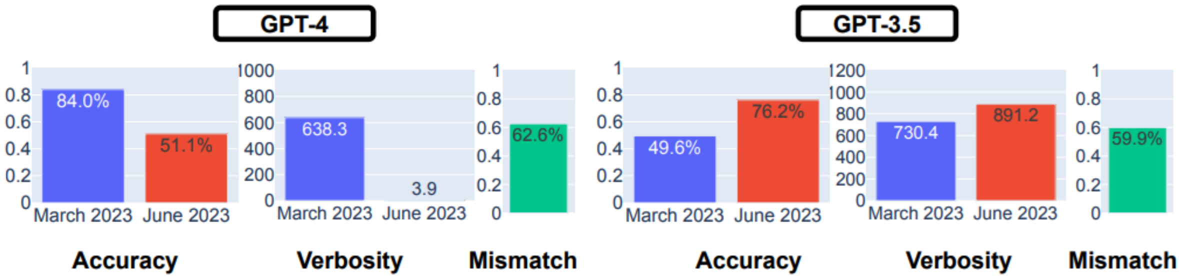


Figure11. Comparison of March and June 2023 versions of GPT-3.5 and GPT-4 for math questions[11]

Should acoustic professionals invest time in learning to code?

While ChatGPT can generate code from natural language input, individuals without coding knowledge will struggle to harness its coding capabilities, at least as of the time of this writing. Paradoxically, in rise of generative AI, learning coding skills are more important than ever. Encouragingly, research has shown that less experienced users stand to gain the most from AI's code generation power, with a substantial increase in productivity by approximately 32%[12]. The investment of time in learning coding skills is becoming increasingly worthwhile and important in today's environment.

Limitations

It is known that OpenAI's GPT like other LLMs has certain limitations, including:

- Toxicity. ChatGPT may sometimes produce responses that could potentially be harmful.
- Hallucinations. It can generate factually incorrect content.
- Infringement of copyright. Users should ensure that they don't inadvertently infringe on copyright issues related to the model's output.

Mitigations for these limitations fall outside the scope of this paper. Nevertheless, awareness of these limitations is essential.

5 CONCLUSIONS

This paper outlined a strategy for leveraging the capabilities of AI for acoustic design, involving the use of LLMs to generate code that can be integrated into software packages. Specifically, it explored the synergy between OpenAI's GPTs and Grasshopper for room acoustic design, with a case study on GPT Python coding to optimise the angles of reflectors for maximising energy projection from 1st order reflections onto the audience plane in Rhino.

The paper tested various OpenAI methods for coding. It was observed that the Custom Instructions feature provided by ChatGPT proved effective in enhancing the quality of the output. OpenAI Playground, powered by GPT-4 and billed according to the token usage for prompts and outputs, occasionally further improved the results. In general, the free service ChatGPT powered by GPT-3.5, produced comparable code quality when provided with effective Custom Instructions.

The experiments illustrated that prompts can significantly influence the quality of the output generated by OpenAI's GPTs. Shortening the chain of reasoning was found to be beneficial. For solving acoustic problems, a successful approach was to translate the physics into the underlying mathematical problem before generating the code. The prompt-for-prompt technique proved useful at times.

Although the code generated by ChatGPT was often non-executable, primarily due to syntax errors, ChatGPT has been demonstrated to be valuable in providing coding strategies, identifying appropriate Grasshopper Python functions for specific tasks, aiding in learning coding languages, assisting in debugging, optimising code, performing code testing, and generating code documentation. It is worthwhile to experiment on fine-tuning OpenAI's GPTs with a substantial and high-quality dataset comprising prompts and manually crafted code for reducing syntax errors.

With the exponential advancement of technology, ChatGPT and generative AI in general, are becoming increasingly powerful co-pilots for professional software. Software interfaces should support coding with generative AI. For acoustic professionals, possessing coding skills can increase productivity more than ever. The investment in time in acquiring these skills is increasingly necessary in today's environment.

6 REFERENCES

1. A. McAfee and E. Brynjolfsson, The second machine age: work, progress, and prosperity in a time of brilliant technologies. New York (2014)
2. N. Komerath, A technology countdown approach to historical timelines. Modified paper presented at the WAVES2020 (2021)
3. McKinsey Global Institute, Notes from the AI frontier – Modelling the impact of AI on the world economy, discussion paper. (September 2018)
4. N. Benaich and the Air Street Capital team, State of AI Report 2023. https://www.stateof.ai/. (12 October 2023)
5. D. F. Carr, Stack Overflow is ChatGPT Casualty: Traffic Down 14% in March. https://www.similarweb.com/blog/insights/ai-news/stack-overflow-chatgpt/ (21 June 2023)
6. Wikipedia, GPT-4. https://en.wikipedia.org/wiki/GPT-4
7. R. Khawaja, Best large language models (LLMs) in 2023, https://datasciencedojo.com/blog/best-large-language-models/# (26 July, 2023)
8. Wikipedia, ChatGPT. https://en.wikipedia.org/wiki/ChatGPT
9. AI Foundations, ChatGPT Update: Custom Instructions in ChatGPT! (Full Guide). https://www.youtube.com/watch?v=TbbA44Jaric (1 September 2023)
10. A. Zhou, et al., Solving challenging math word problems using GPT-4 Code Interpreter with code-based self-verification. Cornell Computation and Language. (15 August 2023)
11. L. Chen, M. Zaharia, J. Zou, How is ChatGPT's Behaviour Changing over Time? https://arxiv.org/abs/2307.09009 (18 July 2023)
12. E. Kalliamvakou, Research: quantifying GitHub Copilot's impact on developer productivity and happiness. https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/ (17 September 2022)