**​​1. GitHub Code Repositories Tested with Links​​**

(1) [ProbableTrain/MapGenerator](https://github.com/ProbableTrain/MapGenerator)

* Random road & building cluster generation
* Closed-loop ecosystem with no custom data input
* Generates pseudo-3D models without height control
* ​**​Result​**​: Irrelevant to project goals → ​**​Abandoned​**​

(2) [Stability-AI/stable-fast-3d](https://github.com/Stability-AI/stable-fast-3d)

* Claims single-image to textured 3D model conversion
* ​**​Practical Issues​**​:
  + High deployment complexity
  + CPU-only rendering limitations
* ​**​Result​**​: Insufficient capability → ​**​Abandoned​**​

(3) [colmap/colmap](https://github.com/colmap/colmap)

* Feature-point scanning for 2D-to-3D conversion
* ​**​Value​**​: Inspired feature recognition approaches
* ​**​Usage​**​: Conceptual reference only (not directly implemented)

(4) [nmoehrle/mvs-texturing](https://github.com/nmoehrle/mvs-texturing)

* Core functionality integrated into data generation pipeline
* ​**​Role​**​: Primary texture processing module

(5) [facebookresearch/pytorch3d](https://github.com/facebookresearch/pytorch3d/tree/main)

* Converts OBJ models into mesh feature representations
* ​**​Role​**​: Critical input processing component

**​​2. Experimental Results Comparison: Initial vs. Optimized Versions​​**

**​​Functional Comparison​​**

| **Feature** | **t1.py (Optimized)** | **test1.py (Initial)** |
| --- | --- | --- |
| Input | OBJ / MTL / TGA | OBJ / MTL / TGA |
| Output | MTL (Kd color) + 256×256 TGA | MTL (Kd color) + 512×512 TGA |
| Primary Use | High-precision texture synthesis | Rapid prototyping |
|  | (Complex texture generation) | (Basic texture prediction) |

**​​Algorithmic Differences​​**

| **Dimension** | **t1.py** | **test1.py** |
| --- | --- | --- |
| Feature Extraction | 33D advanced geometric features | Vertex mean only (3D) |
|  | (Normals, curvature, volume) |  |
| Model Architecture | ✓ Deep GAN (w/ residual blocks) | ✗ Simple MLP |
| Loss Function | ✓ Composite: L1+SSIM+VGG+Style+GAN | ✗ MSE only |
| Training Strategy | ✓ Multi-phase: Warmup+Scheduler | ✗ Basic SGD |
|  | +Early Stopping+Validation |  |
| Data Augmentation | ✓ Color jitter, rotation, flip, blur | ✗ None |
| Output Control | ✓ Tanh [-1,1] + post-processing | ✗ Sigmoid [0,1] |
| Texture Resolution | 256×256 (high quality) | 512×512 (blurry output) |
| Gradient Penalty | ✓ WGAN-GP | ✗ None |
| Caching | ✓ Dataset acceleration | ✗ None |

**​​Module Comparison​​**

| **Component** | **t1.py** | **test1.py** |
| --- | --- | --- |
| Model Class | EnhancedTexturePredictor | MtlTexturePredictor |
|  | + TextureDiscriminator |  |
| Training Function | train\_model\_with\_texture() | Simplified version |
| Generation Function | generate\_mtl\_and\_texture() | Simplified version |
| MTL Processing | ✓ Auto-repair OBJ mtllib/usemtl refs | ✗ Basic write only |

**​​3. Selected Repository & Justification​​**

​**​Chosen Codebase​**​: t1.py

​**​Core Advantages over test1.py​**​:

* ​**​Deeper Architecture​**​: 33D geometric features replace simplistic vertex means
* ​**​Stabler Training​**​:
  + Learning rate warmup
  + Validation-based early stopping
  + Gradient penalty regularization
* ​**​Enhanced Realism​**​:
  + Composite loss (SSIM + VGG + Style + Adversarial)
  + Advanced data augmentation
  + 256×256 high-fidelity textures
* ​**​Production Readiness​**​:
  + Caching mechanisms
  + Exception handling
  + Automatic OBJ material reference fixing

​**​Outcome​**​: Superior detail preservation, structural consistency, and photorealism compared to the rapid-prototyping approach of test1.py.

**​​4. Optimization Approaches Beyond Parameter Tuning​​**

**​​Model Architecture Improvements​​**

* ✓ Implemented U-Net decoder structure
* Potential: Transformer-based feature fusion

**​​Loss Function Enhancements​​**

* Add frequency-domain loss
* Incorporate diversity loss

**​​Training Strategy Upgrades​​**

* Curriculum learning implementation
* Cyclic learning rate scheduling

**​​Feature Extraction Augmentation​​**

* Advanced curvature analysis
* Topological feature extraction

**​​Post-Processing Techniques​​**

* Texture sharpening filters
* Material parameter correction

**​​System-Level Optimizations​​**

* Mixed-precision training
* Distributed data parallel processing

**​​Multi-Category Learning​​**

* ​**​Problem​**​: Model performance degrades with complex datasets (e.g., diverse architectural styles)
* ​**​Solution​**​:
  1. Class-specific style embeddings (office/hotel/residential)
  2. Socioeconomic factor integration (premium vs. budget housing)
  3. Gram matrix weight adjustment (current 20% → 35-40% for style emphasis)
  4. Detail generation trade-off optimization

**​​Overfitting Mitigation​​**

* ​**​Observed in v1.4​**​: Middle-Eastern architecture dataset
* ​**​Countermeasures​**​:
  + Region-specific style normalization
  + Dataset balancing techniques
  + Feature dropout regularization

**​​Parameter Tuning Effects​​**

* Batch size=16: Low learning rate → increased noise artifacts
* Batch size=32 + beta2=0.999:
  + Progressive quality improvement from beta1=0.73 → 0.81 → 0.92
  + Optimal balance between convergence speed and output stability

**​​Summary of Enhancements​​**

| **Dimension** | **test1.py** | **t1.py Enhancements** |
| --- | --- | --- |
| Model Capability | Linear mapping | Non-linear GAN + discriminator |
| Input Information | Crude geometry | Fine geometry + normals + curvature |
| Output Quality | Average textures | Photorealistic outputs |
| Training Robustness | Prone to overfitting | Early stopping + validation |
| Production Viability | Prototype-level | Production-ready pipeline |

​**​**