Implicitus: A Constraint-Compiled Approach to Physical Design

Abstract

Recent advances in text-to-CAD systems have highlighted both the promise and limitations of applying large language models (LLMs) directly to geometric design. While these systems demonstrate early utility in simple part generation, they struggle to scale to real-world physical systems due to the combinatorial complexity and poor structure of existing CAD representations.

This paper proposes an alternative paradigm: a constraint-compiled, field-native design system that bypasses traditional CAD abstractions entirely. Rather than generating geometry directly, this approach represents objects as compositions of continuous fields governed by constraints, enabling a more natural alignment with physics, materials, and manufacturing processes.

We argue that the path to designing complex physical systems is not through scaling geometry generation, but through structuring and composing constraint systems. This paper outlines the architectural foundations of such a system, introduces key primitives, and proposes a roadmap for scaling from simple field-generated objects to complex, multi-domain machines.

---

1. Introduction

The dominant paradigm in digital design today is rooted in parametric CAD systems, which represent objects as discrete features organized into hierarchical assemblies. While powerful, this paradigm suffers from several fundamental limitations:

- Geometry is discrete and brittle
- Design intent is poorly represented
- Optimization is externalized rather than intrinsic
- Multi-physics constraints are difficult to express

Recent efforts to apply LLMs to CAD generation have largely inherited these limitations. Training models on datasets such as ABC and DeepCAD introduces additional challenges:

- Noisy, inconsistent data
- Weak mapping between geometry and function
- Lack of semantic structure

As a result, LLM-generated CAD models often fail to capture the underlying functional and physical intent required for real-world applications.

---

2. Rethinking the Problem

Instead of asking how to generate better geometry, we propose reframing the problem:

> Physical design is the process of resolving interacting constraints into realizable forms.

From this perspective, geometry is not the primary artifact—it is the byproduct of constraint resolution.

This leads to three key shifts:

1. From geometry to fields
2. From features to constraints
3. From assemblies to systems

---

3. Field-Native Representation

At the core of this approach is a field-based representation, such as signed distance fields (SDFs), which describe objects as continuous scalar functions over space.

Advantages include:

- Continuity and smoothness
- Natural support for blending and composition
- Direct compatibility with simulation and optimization
- Alignment with additive manufacturing processes

However, raw field representations alone are insufficient. To scale, they must be structured.

---

4. Constraint-Compiled Geometry

We introduce the concept of constraint compilation:

> A process by which a set of constraints is transformed into a field representation that satisfies those constraints.

Constraints may include:

- Geometric constraints (thickness, curvature, clearance)
- Physical constraints (load paths, energy absorption)
- Interface constraints (mounting points, mating surfaces)
- Manufacturing constraints (printability, material limits)

The system acts as a compiler:

- Input: Constraint graph
- Output: Field representation

---

5. Field Primitives

To enable reuse and composability, we define field primitives as reusable constraint bundles.

Examples include:

- Energy-absorbing lattice region
- Snap-fit interface
- Mounting boss with compliance gradient
- Rotational clearance envelope

Each primitive is defined by:

- A local field transformation
- A set of parameters
- Behavioral guarantees

These primitives serve as the foundational building blocks of the system.

---

6. Constraint Graphs

Rather than organizing designs as hierarchical assemblies, we propose a constraint graph model.

6.1 Structure

- Nodes: Field primitives or regions
- Edges: Relationships between nodes

6.2 Relationship Types

- Spatial alignment
- Motion constraints
- Load transfer
- Exclusion zones
- Interface compatibility

This graph encodes design intent explicitly, allowing the system to resolve interactions globally.

---

7. Hybrid Continuous-Discrete Systems

Real-world systems contain both continuous and discrete elements.

We propose treating discrete components as boundary conditions within the field system.

Examples:

- Bearings
- Fasteners
- Motors
- Electronics

These are represented as:

- Fixed geometries
- Constraint envelopes
- Interface definitions

The system generates continuous material around these anchors.

---

8. Role of LLMs

Rather than generating geometry directly, LLMs operate at the level of intent and system definition.

Responsibilities include:

- Constructing constraint graphs
- Selecting primitives
- Parameterizing behaviors

Example inputs:

- "Design a helmet that absorbs rotational impacts"
- "Create a mount compatible with a GoPro interface"

The LLM acts as a planner, while the constraint compiler resolves the design.

---

9. Incremental Path to Complexity

We propose a staged approach to scaling system complexity:

Level 1: Pure field-generated objects

- Helmets
- Lattice structures

Level 2: Field + static inserts

- Mounts
- Enclosures

Level 3: Field + compliant mechanisms

- Flexures
- Hinges

Level 4: Field + electromechanical integration

Level 5: Full systems

Each stage builds on the previous, enabling gradual expansion of capability.

---

10. Application Domains

Initial target domains should exhibit:

- High geometric complexity
- Strong dependence on material behavior
- Minimal discrete components

Examples include:

- Protective equipment
- Footwear
- Medical devices
- Lightweight structural components

These domains provide immediate value and tractable complexity.

---

11. Implications

This approach enables:

- Continuous design spaces
- Intrinsic optimization
- Physics-aware geometry generation
- Software-defined manufacturing

It represents a shift from designing objects to defining systems.

---

12. Conclusion

The future of physical design will not be driven by better geometry generation alone. Instead, it will emerge from systems that can interpret, compose, and resolve constraints into functional forms.

By embracing field-native representations and constraint compilation, we can build a new design stack that aligns more closely with the realities of physics, materials, and manufacturing.

This paradigm does not replace CAD—it transcends it.

---

13. Future Work

Key areas for further development include:

- Formal definition of constraint languages
- Efficient solvers for large-scale constraint graphs
- Integration with simulation pipelines
- Tooling for primitive authoring
- Interfaces between LLMs and constraint systems

The opportunity is not incremental—it is foundational.