RSCE Codex - Phase 1 - rsce.py (ASC2 Clean)

rsce_frequency_gate_patch.py

```
from weights import compute_weight
from seed_recovery import find_seed
import math

class RSCEngine:
    def __init__(self, base_radius=1.0, tension_k=1.0, freq_gate=888.0, threshold=0.1):
        self.base_radius = base_radius
        self.tension_k = tension_k
        self.freq_gate = freq_gate
        self.threshold = threshold

    def map_to_axes(self, observed):
        positions = {"H1": (1, 0), "H2": (-1, 0), "O": (0, 0)}
        return positions

    def frequency_check(self, observed):
        element_frequency_map = {
            "H": 888.0,
            "O": 432.0
        }
        tokens = observed.replace("+", "").split()
        for token in tokens:
            element = token[-1]
            if element in element_frequency_map:
                freq = element_frequency_map[element]
                if abs(freq - self.freq_gate) > 100:
                    return False
        return True

    def tension(self, p1, p2):
        dx = p1[0] - p2[0]
        dy = p1[1] - p2[1]
        distance = math.sqrt(dx**2 + dy**2)
        return self.tension_k * distance

    def mirror_axes(self, p, axis):
        if axis == "x":
            return (-p[0], p[1])
        elif axis == "y":
            return (p[0], -p[1])
        return p

    def full_axes(self):
        return ["x", "y"]

    def execute(self, observed):
```

RSCE Codex - Phase 1 - rsce.py (ASC2 Clean)

```
if not self.frequency_check(observed):
    raise ValueError("Slice out of resonance gate")

positions = self.map_to_axes(observed)
full = set(positions.values())
weights = {}

for p in list(full):
    for axis in self.full_axes():
        mirror_p = self.mirror_axes(p, axis)
        if self.tension(p, mirror_p) < self.threshold:
            full.add(mirror_p)
            weights[mirror_p] = compute_weight(mirror_p, self.base_radius)

seed = find_seed(full)
return seed, full, weights
```