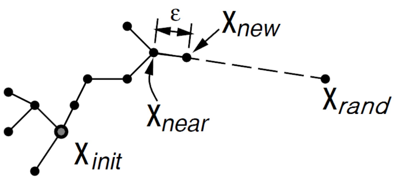
RRTs (Rapidly - exploring Random Trees)

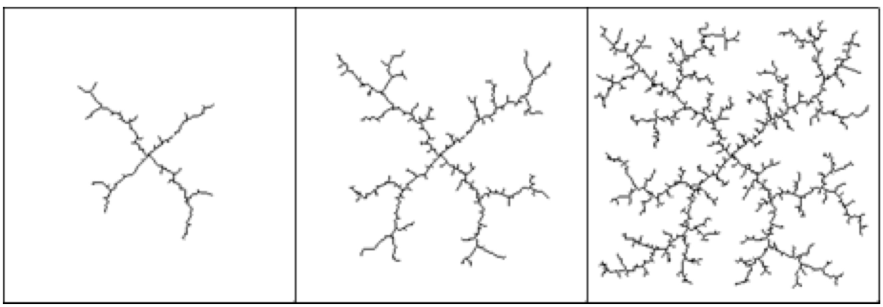
→ Data structure & path planning algorithm, designed for efficiently search paths in non-convex high-dimensional spaces.

\# vertices of tree.

```
GENERATE_RRT(x_init, K, Δt)
    T.init(x_init);                         └ initial pose of robot.
    for k = 1 to K do                       (loop can change to checking closest distance from tree to goal)
        x_rand ← RANDOM_STATE();            can change to RANDOM_FREE_STATE
        x_near ← NEAREST_NEIGHBOR(x_rand, T);   x_near ∈ T.
        u ← SELECT_INPUT(x_rand, x_near);   Control (input)
        x_new ← NEW_STATE(x_near, u, Δt);
        T.add_vertex(x_new);
        T.add_edge(x_near, x_new, u);       ← Can check obstacle.
    Return T
```



- In practice, "goal bias" is required : replacing Xrand into Xgoal for pre-fixed frequency.



↳ Example of RRT expansion

RRT*

↳ stands for optimal.

Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**

      Cardinality, |V| Constant in steering func.

    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;

      r(card(V)), distance.

    $V \leftarrow V \cup \{x_{\text{new}}\}$;

      cost from $X_{init} \to x_{nearest}$ / Straight path.

    $x_{\text{min}} \leftarrow x_{\text{nearest}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;

      line cost.

    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $\boxed{E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\};}$
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;