**InfiniTAM v3 : A Framework for Large-Scale 3D Reconstruction with Loop Closure**

**Ref.** [**https://www.groundai.com/project/infinitam-v3-a-framework-for-large-scale-3d-reconstruction-with-loop-closure/**](https://www.groundai.com/project/infinitam-v3-a-framework-for-large-scale-3d-reconstruction-with-loop-closure/)

1. Introduction

InfiniTAM系統的第三個版本。它的目標在縮小KinectFusion系統背後的理論與我們InfiniTAM軟件包中的實際軟件實現之間的差距。

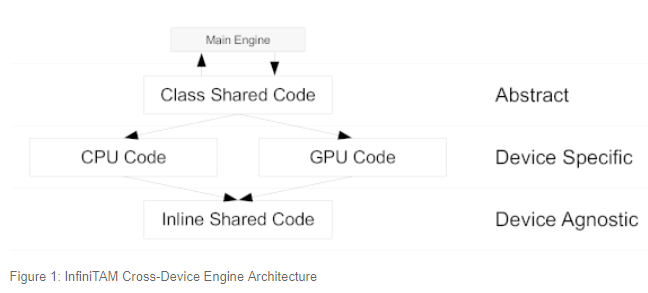
* 1. What’s New

與以前版本的InfiniTAM相比，我們增加了一些令人興奮的新功能，並對低級代碼進行了大量改進，從而顯著提高了我們的相機追踪性能。 我們期望最感興趣的新功能是：

1. 強大的相機追踪模組。 我們在InfiniTAM v2中發布的追蹤器之上構建，使系統更加強大。 新的追踪器允許通過使用場景的光線投影對齊深度圖像以及連續RGB幀的對齊來估計相機姿態，以更好地估計存在幾何表面不良的相機姿態。 此外，我們實現了一個追蹤質量評估系統，可以檢測到追踪失敗，從而觸發下一個描述的重新定位系統。
2. Glocker等人的keyframe-based random ferns camera relocaliser的實現。 這使在追蹤失敗時恢復相機姿態。
3. 一種基於全局一致TSDF重建的新方法，其基礎是將場景劃分為子圖，並優化它們之間的相對位置以構建一致的總體地圖。
4. Keller等人基於surfel的重建方法的實現。Surfels是場景重建中TSDFs的一個有趣替代方案。 一方面，它們可以用於處理動態場景，因為surfel模型的表面可以移動，而不需要更新周圍的截斷區域。 另一方面，在使用TSDF的情況下，可見性確定和surfel場景中的碰撞檢測的成本要高得多。 我們添加到InfiniTAM的實施旨在讓其他人在單一框架內輕鬆探索這種折衷。
5. Cross-Device Implementation Architecture

我們的實現遵循責任鏈設計模式(chain-of-responsibility design pattern)，其中數據結構（例如ITMImage）在多個處理引擎（例如ITMTrackingEngine）之間傳遞。 引擎理想情況下是無狀態的，每個引擎負責整個處理管道的一個特定方面。 狀態通過包含已處理信息的對像傳遞。 最後，一個父類（ITMMainEngine）擁有所有對象的實例，控制引擎的流程。

每個引擎進一步分為3層，如圖1所示。最上面的所謂的抽象層被庫的主引擎訪問，通常只是一個抽象接口，儘管此時可能會共享一些通用代碼。 該抽象接口在下一個設備特定層中實現，該特定層在例如 一個CPU和一個GPU實現。 使用例如 OpenMP或其他硬件加速體系結構是可能的。 在第三個設備不可知層(Device Agnostic Layer)，有一些內聯C代碼可以從更高層調用。



例如追蹤引擎，抽象層可以包含用於通用優化錯誤函數的代碼，設備特定層(Device Specific Layer)可以包含循環或CUDA內核調用以評估圖像中所有像素的錯誤函數，並且設備不可知 Layer包含一個簡單的內聯C函數來評估單個像素中的錯誤。

請注意，在InfiniTAM v3中，源代碼文件已更改位置。 InfiniTAM v2中有一個文件夾包含Device Agnostic和Device Specific文件，我們現在為每個模塊（追踪，可視化等）提供單獨的文件夾，每個文件夾分為抽象的，特定的和不可知的。

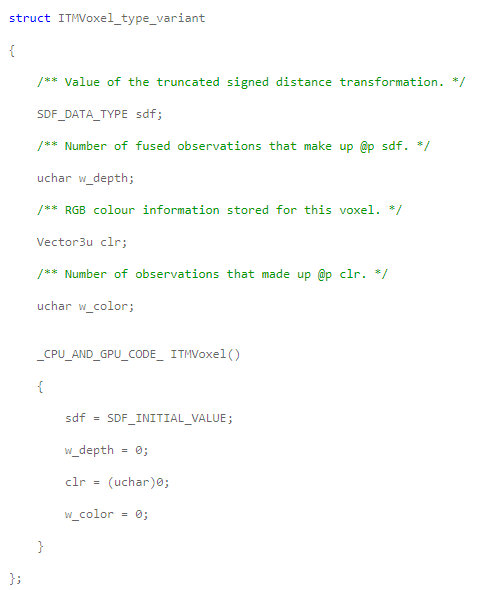
1. Volumetric Representation with Hashes

允許InfiniTAM將KinectFusion擴展到大型3D環境的關鍵組件是使用哈希查找(hash lookup)的體積表示。

實現這種表示的數據結構和相應的操作如下：

* 體素塊陣列(Voxel Block Array)：保存融合的顏色和3D深度信息
* 雜湊表和雜湊函數(Hash Table and Hashing Function)：可以快速訪問的體素塊陣列
* 雜湊表操作：插入，檢索和刪除
  1. The Voxel Block Array

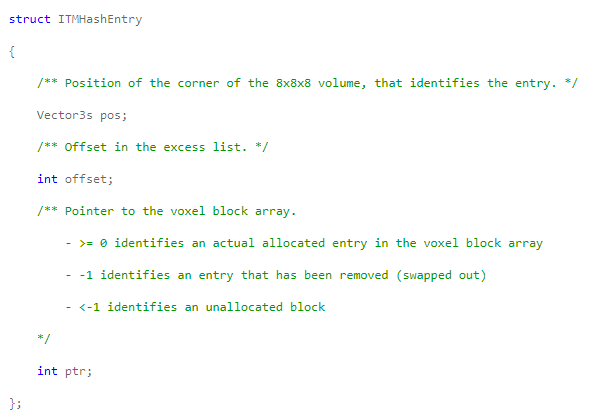
深度和顏色信息保存在ITMVoxel物件



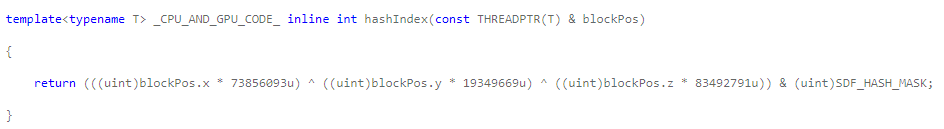
體素按預定義大小的塊分組（目前定義為8×8×8）。 所有的塊都被存儲為一個連續的數組，以後稱之為體素塊陣列或VBA。在實現中，這有一個2 ^ 18元素的定義大小。

* 1. The Hash Table and The Hashing Function

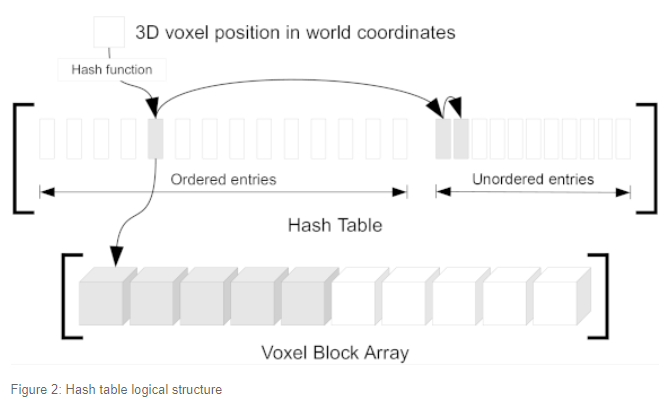
為了快速有效地找到體素塊陣列中某個體素塊的位置，我們使用一個哈希表。 該雜湊表是以下形式的ITMHashEntry對象的連續陣列：



用於定位哈希表的條目的哈希函數hashIndex採用3D體素塊的角坐標blockPos併計算如下的索引

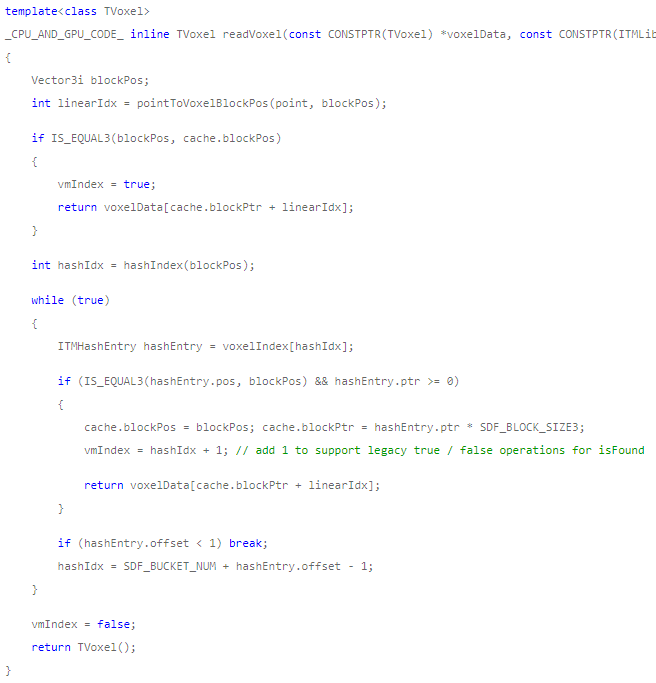


為了處理雜散衝突，每個雜散索引指向一個大小為1的存儲桶，我們認為這是雜湊表的有序部分。有一個額外的無序的額外清單，一旦有序的存儲桶被填滿，將被使用。無論哪種情況，哈希表中的每個ITMHashEntry都會在體素塊陣列中存儲一個偏移量，因此可以用於為每個特定的體素塊定位體素數據。這個總體結構如圖2所示。



* 1. Hash Table Operation

使用哈希表時使用的三個主要操作是插入，檢索和刪除條目。 在當前版本的InfiniTAM中，只提供前兩個，目前尚未要求或執行刪除操作。 檢索操作使用的代碼如下所示：



通過存儲在雜散表中的鍊錶(link list)的元素來插入和檢索。給定世界坐標中的目標三維體素位置，我們首先通過將體素位置除以體素塊陣列的大小來計算其相應的體素塊位置。接下來，我們調用雜散函數hashIndex來從雜散表的有序部分計算存儲區的索引。 然後檢查存儲桶中的所有元素，檢索目標塊位置並插入未分配的雜散條目。如果找到了，檢索將返回存儲在雜散條目指定的塊內目標位置的體素。插入在體素塊陣列內保留塊並用包含保留體素塊陣列地址和目標塊3D世界坐標位置的新條目填充雜散表。

如果存儲桶中的所有位置都耗盡，則鍊錶的列舉(enumeration)將移動到散雜散表的無序部分，並使用偏移提供下一個雜散條目的位置。當發現偏移量小於或等於-1時，列舉結束。此時，如果仍未找到目標位置，則檢索返回空體素。插入在雜湊表的無序部分和體素塊陣列內的塊中保留未分配的條目，用包含保留體素塊陣列地址和目標塊3D世界坐標位置的新條目填充雜散表將鏈接列表中上次找到的條目中的偏移量字段更改為指向新填充的條目。

用於雜湊表的無序部分和用於體素塊陣列的保留操作使用預先填充的分配列表，並且在GPU代碼中使用原子操作。

所有哈希表操作都是通過這些函數完成的，並且沒有直接內存訪問被當前版本的代碼鼓勵或確實允許。

1. Method Stage



如圖3所示，來自攝像機的每個輸入幀都由類似於中的流水線和原始KinectFusion系統處理。ITMMainEngine是包含流水線入口點的抽像類別，形式為 ProcessFrame方法。有兩種具體的流水線實現：ITMBasicEngine，一種類似於中描述的標準融合算法，以及ITMMultiEngine，允許生成全局一致的重建（InfiniTAM v3中的新特性），對管道的ProcessFrame方法的每次調用都可以在六個不同的階段進行邏輯分割，每個階段都使用一個或多個處理引擎實現：

* Tracking：通過將當前深度圖像與先前幀的世界模型投影進行擬合來獲得新幀的相機姿態。這是使用ITMTracker，ITMColorTracker，ITMDepthTracker和ITMExtendedTracker引擎實現的。
* Allocation：基於深度圖像，根據需要分配新的體素塊並建立所有可見體素塊的列表。 這是在ITMSceneReconstructionEngine（Voxel map）和ITMSurfelSceneReconstructionEngine類（surfel map）中實現的。
* Swapping in：如果需要，地圖數據將從主機內存交換到設備內存。 這是使用ITMSwappingEngine實現的。 當前版本僅支持具有體素的BasicEngine。
* Integration：當前的深度和顏色框架集成在地圖中。此操作再次在ITMSceneReconstructionEngine（voxels map）和ITMSurfelSceneReconstructionEngine（surfel map）類中執行。
* Raycast：世界模型從當前的姿勢渲染以供可視化目的使用，並在下一幀的追踪階段使用。 這使用ITMVisualisationEngine。ITMMultiVisualisationEngine和ITMSurfelVisualisationEngine類。
* Swapping out：映射中不可見的部分從設備內存換出到主機內存。 這是使用ITMSwappingEngine實現的。 當前版本僅支持具有體素的BasicEngine。

主要處理引擎包含在ITMLib命名空間中，以及ITMLowLevelEngine，ITMViewBuilder，ITMMeshingEngine和ITMMultiMeshingEngine。 它們分別用於低級處理（例如圖像複製，梯度和重新縮放），圖像準備（將深度圖像從無符號短到浮點值，正常計算，雙邊濾波轉換）以及通過Marching-Cubes算法生成網格[7]。

圖像採集程序依賴於大量的輸入傳感器，可以在InputSource命名空間中找到，而relocaliser實現包含在FernRelocLib命名空間中。

最後，主UI和關聯的應用程序包含在InfiniTAM命名空間中。

* 1. Tracking

在追踪階段，我們必須根據3D世界模型來確定新圖像的姿態。我們可以基於ITMDepthTracker的新深度圖像或基於具有ITMColorTracker的彩色圖像來執行此操作。從InfiniTAM系統的第3版中，我們還提供了一個修訂和改進的跟踪算法：ITMExtendedTracker。 所有擴展抽象ITMTracker類，並具有在CPU和CUDA上運行的特定於設備的實現。

ITMDepthTracker：

* 從初始猜測的角度渲染表面點的地圖V和表面法線的地圖N。
* 將深度圖像中的所有點p投影到V和N中的點上，併計算它們與平面的近似表面之間的距離。
* 通過求解線性方程組來尋找R和t來最小化平方距離之和
* 迭代前兩個步驟，直到收斂。

在我們的實現中使用深度圖像的分辨層次來改善收斂行為。

ITMColorTracker：

* 從初始猜測的角度創建表面點的列表V和顏色的相應列表C。
* 將V中的所有點投影到當前的彩色圖像I中，併計算顏色的差異
* 使用Levenberg-Marquardt優化演算法查找R和t最小化平方差的總和。

再次使用彩色圖像中的分辨率層次結構，並將表面點列表以4倍進行二次採樣。

ITMExtendedTracker：

ITMExtendedTracker類別允許對齊體感器捕獲的當前RGB-D圖像

更具體地說，在其默認配置中，ITMExtendedTracker的一個實例可以無縫地替換ITMDepthTracker，同時提供通常更好的跟踪準確性。 與簡單的基於ICP的跟踪器的主要區別如下：

* 在計算與輸入深度圖像的每個像素相關的誤差項時，Huber-norm部署不是相當標準的L2範數。
* 深度圖像的每個像素的誤差項除了受到剛描述的魯棒標準的影響之外，還根據其由傳感器提供的其深度測量來加權，以便考慮與遠點相關的距離測量的噪聲性質 遠離相機（權重隨距離讀數的增加而下降）。
* 其中點p是V中的投影與距表面的平面近似的距離大於可配置閾值，在錯誤函數中被認為是最小化的，因為它們可能是異常值。
* 最後，通過SVM分類器評估ICP優化階段的結果（內部像素的百分比，Hessian行列式和殘差總和的行列式），以便在跟踪成功和失敗之間分離，以及確定成功結果是好還是不好 準確性。 在追踪失敗的情況下，重新定位器被激活以嘗試估計當前的相機姿態並恢復定位和映射循環。

我們用每像素光度誤差項來增大上述擴展跟踪器的幾何誤差項。更詳細地說，我們試圖最小化當前RGB圖像中的每個像素與體感器捕獲的前一幀中的對應像素之間的強度差異（注意，不同於上一節中描述的ITMColorTracker，我們不需要有權訪問顏色重建，因為我們在匹配階段不使用體素顏色）。 幀到幀的像素強度差異估計如下：

* 從當前RGB-D對中的像素關聯的強度和深度值從圖像中被採樣（每個強度值I被計算為RGB顏色通道的加權平均值：

I =0.299R+0.587G+0.114B）

* 通過根據相機內部參數反投影其深度值來計算當前相機參考幀中的像素的3D坐標。
* 給定體感器姿態（初始猜測也用於計算幾何誤差項），將3D像素坐標引入場景坐標。
* 使用前一幀的估計姿態將像素坐標恢復到上一個相機的參考坐標。
* 將這些坐標投影到先前的RGB圖像上，並通過雙線性插值(bilinearly interpolating)與其相鄰像素相關的值來計算採樣點的強度。

如前所述，受到基於深度的加權和魯棒損失函數（在這種情況下，我們使用Tukey's）的每個像素強度差異的總和被添加到上述幾何誤差項，使用適當的比例因子， 以說明值的不同範圍（在當前實施方式中為0.3）。 梯度和Hessian值也結合在一起。 然後Levenberg-Marquardt優化演算法以最小化誤差並估計最終姿態。

與其他追蹤器一樣，計算分辨率層次結構，並從最粗糙的分辨率開始執行相機對齊，以改善收斂行為。

Configuration：

ITMLibSettings中的字符串允許選擇使用哪個追踪器，以及指定追踪器特定的配置值。默認情況下，創建一個ITMExtendedTracker的實例（禁用色彩跟踪能量項 - 以允許使用Occipital的Structure1等深度傳感器）。其他配置字符串可以在ITMLib/Utils/ITMLibSettings.cpp中找到。

所有跟踪器實現都只使用該設備（在CUDA特定的子類中，依賴於CPU），僅用於計算函數，梯度和Hessian值。 其他一切，比如優化，都是在主抽像類別中完成的。

* 1. Allocation

分配階段分為三個獨立的部分。我們的目標是盡量減少阻塞操作的使用，並完全避免使用關鍵部分。

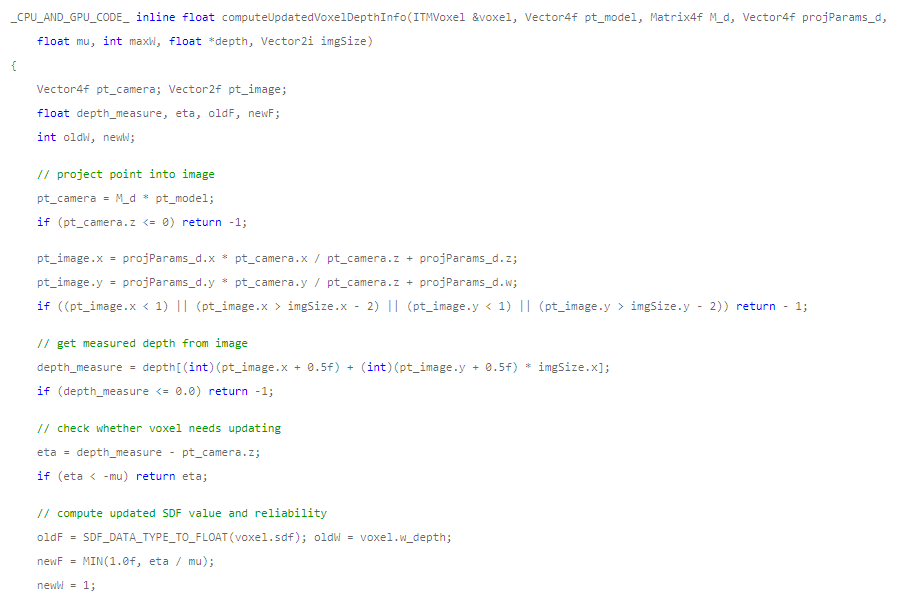
首先，對於來自深度圖像的每個2.5D像素，我們反向投影連接d-μ到d +μ的線，其中d是圖像坐標中的深度，並且μ是固定的可調參數。這導致世界坐標中的一條線，其與多個體素塊相交。我們為每個這些塊搜索雜散表，並為每個未分配塊查找一個免費的雜散條目。這些存儲在下一個分配階段，使用兩個數組，每個數組與哈希表的元素數相同，包含有關新哈希條目的分配和可見性的信息。請注意，如果同一深度圖像的兩個或更多塊被映射到相同的雜散條目（即，如果我們有幀內雜散衝突），則只會分配一個。由於幀內相機運動相對較小，此影像自動固定在下一幀。

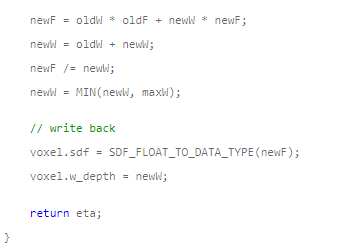
其次，我們為先前構建的分配和可見性數組中的每個非零條目分配體素塊。這是通過在一堆自由體素塊索引上使用原子減法完成的，即我們將可用塊的數量減1，並將堆棧的前一個頭添加到雜散項。

第三，我們建立一個實時哈希條目列表，即包含在可見的平截頭體內投影的列表。這稍後將被集成和交換進入/退出階段使用。

* 1. Integration

在整合階段，來自最新圖像的信息被納入3D世界模型。這與原始KinectFusion算法基本相同。對於來自上方的任何可見體素塊中的每個體素，調用函數computeUpdatedVoxelDepthInfo。如果體素位於新深度圖像中觀察到的表面之後，則該圖像不包含任何有關它的新信息，並且函數返回。如果體素靠近觀察表面或在觀察表面的前方，則將相應的觀察結果添加到累積和。這在下面的函數computeUpdatedVoxelDepthInfo的列表中進行了說明。





與原始KinectFusion集成步驟的主要區別在於，上述更新函數不是針對固定體積中的所有體素而是僅針對當前標記為可見的塊中的體素。